129

A SOFTWARE PLANNING AND DEVELOPMENT METHODOLOGY WITH RESOURCE ALLOCATION CAPABILITY

A dissertation

by

JOSEPH BRUCE MICHELS

Submitted to the Graduate College of
Texas A & M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 1986

Major Subject: Industrial Engineering

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD-A186 088

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFIT/CI/NR 87-129D	2. GOVT ACCESSION NO.: AD-A186 088
3. RECIPIENT'S CATALOG NUMBER: DTIC FILE COPY	
4. TITLE (and Subtitle): A Software Planning And Development Methodology With Resource Allocation Capability	5. TYPE OF REPORT & PERIOD COVERED: THESIS/DISSERTATION
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Joseph Bruce Michels	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: AFIT STUDENT AT: Texas A&M Univ	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: AFIT/NR, WPAFB OH 45433-6583	12. REPORT DATE: 1986
	13. NUMBER OF PAGES: 140
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC ELECTE S D NOV 23 1987 C D

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE: IAW AFR 190-1

LYNN E. WOLAVER 20 Oct 87
Dean for Research and
Professional Development
AFIT/NR

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

ATTACHED

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

87 11 10 092

-A-

129

ABSTRACT

A Software Planning and Development Methodology with Resource Allocation Capability. (December 1986)

Joseph Bruce Michels, B.S., Weber State College;
M.S., University of Southern California

Chair of Advisory Committee: Dr. Leland T. Blank

The purposes of this research are to examine the planning and development of large, generic based software development projects and to propose a structured methodology for software planning and development (SPD). The SPD methodology presented parallels the classic software life cycle protocol and can be used in conjunction with this life cycle model to structure decision making and resource planning in a variety of environments. The concepts of kernel construct and "templates" are developed in conjunction with a resource planning and allocation procedure. Three distinct elements are integrated throughout the proposed methodology: Technical, Managerial, and Resource Use determination. Manpower loading relationships are developed to devise an allocation scheme for different personnel resources required within the software development effort.

The generic kernel and associated templates are shown to apply to virtually any type of software development environment. A major software development project is comprised of many different generic kernels, each one representing a different function or routine within the software design. Various templates are used to tailor the kernels to a specific application environment. This tailoring facilitates graphical depiction of the necessary interconnections, databases, and protocols to be identified.

Cost estimation based on resource use is determined for in-house personnel development and external contract vendor development. A building block approach to cost estimation is presented. Each block represents a specific development step of the SPD methodology for software design and development. A personnel resource allocation matrix (PRAM) is designed, which shows the relationships between the personnel resource types, the personnel types available in-house for each step, and the requirements for external contract support. The total personnel cost of the project can be estimated from the various entries of the PRAM, broken down by quantity of in-house and contract support. The fiscal and equipment resources are discussed in the SPD methodology and the cost estimation for personnel is modeled.

All examples presented are based upon a manufacturing scenario; however, the methodology is applicable to any type of software development activity.

A SOFTWARE PLANNING AND DEVELOPMENT METHODOLOGY WITH RESOURCE ALLOCATION CAPABILITY

A Dissertation

by

JOSEPH BRUCE MICHELS

Approved as to style and content by:

Leland T. Blank
Leland T. Blank
(Chair of Committee)

Sallie Sheppard
Sallie V. Sheppard
(Member)

Donald R. Smith
Donald R. Smith
(Member)

Milton J. Fox
Milton J. Fox
(Member)

G K Bennett
G. Kemble Bennett
(Head of Department)

December 1986

DEDICATION

To my twin brother, Dave, whose untimely death during this endeavor caused me to realize what is the true value of life.

ACKNOWLEDGEMENTS

There are many people who must be acknowledged for the support they provided during my graduate program at Texas A&M.

Dr. Lee Blank, my Committee Chairman, must be commended for the tremendous support, guidance, and direction he so aptly provided.

The enthusiasm of my committee, Dr. Sallie Sheppard, Dr. Bob Fox, Dr. Don Smith, and the graduate college representative, Dr. Eugene Sander, were instrumental in providing a strong support base in which to carry out the necessary research activities.

The sage, wise, and often sought counsel of Dr. Horace Van Cleave was important. Dr. Van Cleave provided support at times when it was needed the most.

Major General M. Gary Alkire and Lieutenant Colonel David J. Hewer were instrumental in allowing me to go back to school. Without their support, this work would not have been possible.

Mary Dudic of the Honeywell Corporation provided the needed manpower loading relationships data. Her help was invaluable in developing the personnel resource allocation model.

My parents must be recognized for providing the moral support and encouragement during the many trying times when I wondered why certain things happen.

Colonel and Mrs. Henry Hill were instrumental in providing the support I needed after the death of my brother. Their caring will never be forgotten.

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Background on Information and Automation

With the introduction of the computer into virtually all facets of enterprise and modern life, information and information flow are vital to the success of any organization. Without the receipt of timely information, decision makers are at a serious disadvantage as contrasted with their competition when key managerial decisions are required. Computer systems that are fully integrated allow for the access of various data types within the enterprise. This integration allows decision makers real time data access that otherwise would not be available. Software is a key bridging link for information integration between people, geographic locations, technologies, and processes. With full information integration between these elements, better managerial decisions are made. The end result is an increase in productivity and overall cost effectiveness of the enterprise operation.

In order for information integration to be effective in a corporate erterprise, careful planning must be

---

This dissertation follows the format used in IEEE Tranactions on Software Engineering.

accomplished in the early developmental stages of overall system design. The life cycle approach to system development allows the designer to analyze all of the elements of the system and study the various interactions which affect each element before any system component is actually developed. In this process particular factors which may affect overall software systems development must be identified early in the design and development stages.

The development of an integrated information management system is valuable in many different sectors of business. As is well known, one of the sectors in which significant economic return, productivity enhancement, and quality improvement can be demonstrated is manufacturing. Manufacturing is a prime example because of the many high technology systems which involve computers and information processing systems that control manufacturing equipment. People, production equipment and software are closely linked in the manufacturing environment. Improved information processing in all three elements enhances overall efficiency.

Automation can be defined to be the "application to established industrial processes of artificial devices which can simulate the human psychic functions (senses, memory, standards, intelligence) in order that these

processes may acquire characteristics of adaptability and self-optimization" [25]. An essential feature of this definition is its implication that somewhere in a physical process, or its control system, there exists data which plays a vital role. In fact, much of the activity in a manufacturing system concentrates upon data, rather than the actual material processing activities as emphatically pointed out in the following statement made by Mr. James Lardner, Director of Manufacturing Operations, John Deere Company [45]:

> "The principal activity of most people engaged in manufacturing is creating, analyzing, transmitting, and managing data, while the actual material transformation is a secondary activity."

If information is defined as the data that is playing the vital role, some type of control function is required which insures that data coordination is totally effectuated. Software must provide much of the control for total information systems integration in an automated environment.

The driving economic factor for manufacturing automation is software development [39]. Industry estimates that software is the determining cost factor of over 90% of the total cost of an automation project [9]. Manufacturing has historically, except for the last five to ten years, been a definite strength of the

American economy. However, that strength has been severely eroded by foreign manufacturers due in large part to their effective introduction and use of automated manufacturing methods in the workplace here and in overseas companies.

The factories of America are often disjointed, disorganized, and lack the total integration of all the manufacturing functions [47]. This disorganization has allowed the various manufacturing activities (design, production scheduling, inventory control, machining, forming, and assembly) to develop and operate unique types of hardware, software, and databases for each discrete activity. This development has become characteristically known by the term 'islands of automation'. None of the various functions are integrated to capitalize on the synergistic effect present in an overall systems design [24]. The result is that many factories have little information system integration, yet, they have many systems developed over time and for many excellent, but poorly coordinated programs.

Systems Engineering

A system is a set of interrelated components working together toward a common goal or objective.

Systems are composed of components, attributes, and relationships. Relationships may be described as follows [8]:

1) Components are the operating parts of a system consisting of input, process, and output.
2) Attributes are the properties or discernible manifestations of the components of a system. These attributes characterize the parameters of a system.
3) Relationships are the links between components and attributes.

In the context of software development, the system can be considered to be comprised of two characteristic elements--the managerial and the technical. Diametric opposition of these two elements usually occurs when the managerial elements have specific constraints on time, budget, and degree of technological level thought to be necessary to achieve the desired design function. The technical element, on the other hand, usually has little consideration of these problems; rather, it attempts to define the degree of technological sophistication which the project should develop. The result, in many cases, is competing project objectives which do not easily lead to overall project success.

The systems engineering approach to technology management provides an environment for the integration

of both elements. Systems engineering is a process that has been recognized to be essential in the orderly evolution of man-made systems. It involves the application of efforts necessary to (1) transform an operational need into a description of system performance parameters and a preferred system configuration through the use of an iterative process of functional analysis, definition, design, synthesis, optimization, test and evaluation; (2) integrate related technical parameters and assure compatibility of all physical, functional, and program interfaces in a manner that optimizes the total system definition and design; and (3) integrate performance, productibility, reliability, maintainability, supportability, and other specialities into the total engineering effort.

Need for a Comprehensive Planning Methodology

An understanding of the software requirements as specified in any system specification indicates that a method is required which assists in planning for information synthesis and integration in manufacturing systems software. When the software is utilized to operate and control the manufacturing activities, information and decision integration will be present in the overall manufacturing system of the enterprise.

This system should provide cost estimation and resource allocation models which can track cost accumulation throughout the software system life cycle. The model itself should be based on factors more encompassing than simply lines of software code, and it should be able to identify the significant phases of the life cycle in which substantial resources are expended. The model should be able to be accessed at various parts of the overall systems life cycle and should possess the capability to graphically represent the various components, attributes, and relationships of a system. The design and development of such a methodology is the result of the research presented in this dissertation.

The purposes of this research are to examine the planning and development of large software projects and to use the systems engineering approach to develop a structured methodology for software planning and development (SPD) activities. It is conceived that such a structure can be applied in a variety of working environments i.e., defense, manufacturing, services, health care, etc. The end product of this research effort is a documented structure including a graphics-based resource allocation model that can be applied to software development personnel assignments.

Software Planning and Development (SPD) Methodology Overview

The systems engineering approach provides a global perspective of any problem and allows the analyst the ability to systematically define, design, develop and operate systems by taking into account all aspects of the environment and the system itself from the perspective of the management and technical decision points [9]. Good system design reflects an optimum balance among performance, support, and economic factors which is attained through a trade-off and analysis effort accomplished in the early stages of the system development [7].

The systems approach in the development of software cost estimation allows an analyst to understand the various interactivities that the software has and the equipment that must be interconnected. The developed methodology employs two distinct elements--one technical, the other managerial. The technical element addresses the various technical factors that are required for software development. These factors include system connectivity, data protocols, use of different types of equipment, dependence on various data types, and the like. The managerial element considers proper staffing of the development team, proper

organizational placement in the overall corporate hierarchy of the development team, and definition of the support environment in which the software is developed. The support development environment includes the components of people, hardware, and software productivity enhancement tools.

Many senior decision makers organizationally responsible for the development and implementation of software projects do not have the technical background required to fully understand the various aspects of software development and the different amounts of resources required to adequately complete a software development project. The significant problem with the introduction of Computer Integrated Manufacturing (CIM) is a question of managerial acuity rather than technical diligence [17]. The result in many cases is an industrial software development project which does not adequately achieve all of the initial design objectives and has a final cost overrun of four to six times the initial cost estimate [87].

An organizational and cost modelling approach is required which clearly illustrates how costs are accumulated during software systems development life cycle. This modelling approach should provide strategic management a more responsive method to

allocate resources and to improve managerial decision making.

Many of the available software cost estimation models, for example, RCA Price S, Jensen, COCOMO, and Putnam [5], fail to address the overall cost estimation/resource allocation question from a strategic management perspective. The result in most cases is a model that is typically based upon lines of developed code. The level of accuracy available from these models may be sufficient as an intermediate step in the overall cost estimation/resource allocation process; however, these models may fail to provide strategic level management with a good understanding of how the overall cost estimates and resource allocations are achieved. Thus, in a global sense, these models are possibly not 'robust' enough to adequately track costs incurred throughout the entire system life cycle.

The system life cycle begins with the initial identification of a need and extends through planning, research, design, production or construction, evaluation, consumer use, field support, and ultimate product retirement [7]. A model is required which addresses cost accumulations in all these phases for varying software development environments.

Characteristics of the Developed Methodology

This research has resulted in a software planning and development methodology which possesses project organizing, resource identification, and cost estimation. Technical and managerial elements important to software development are addressed and integrated into the developed methodology. Development by in-house personnel and external contract vendors is considered. The methodology is equally suited for use in a variety of applications that address government, business, or commercial requirements. The examples used in this research are focused on manufacturing. A resource allocation formulation is presented for allocation of three types of resources present in a software development project: (1) personnel, (2) equipment, and (3) fiscal. The concept of a design kernel is presented. A kernel, as used in this context, is defined to be a set of functions necessary for accomplishing a particular task. Generic kernels are developed for different functional applications and then detailed for specific software needs. This approach to software planning and development provides a global viewpoint of the project or list of projects. The various interactions, connections, common data elements, interfaces, and protocols can all be graphically

displayed. This type of display identifies most incongruities in the design phase before the system is actually implemented.

The SPD methodology focuses on the method in which software development programs are managed and how various resource categories are allocated in a software development project. The methodology does not consider the various types of information that are coded in the software, methods of particular data coding, information data structures, data scheduling or handling techniques, or the quality or richness of the information in the system. The SPD methodology could be extended to include some or all of these components as separate research efforts. Unlike the SPD methodology, however, these extensions would not be generic as they would require domain specific information.

A variety of templates is designed to examine certain development actions of a software project. These development actions are important from a strategic management perspective and are considered to be critical to the overall success of such a project. Each template is a flow-chart-style document which addresses critical development questions. These templates, along with the kernels, form the crucial parts of this methodology. Although the various templates are generic in design and can be transported to a variety of application unique

environments, this work addresses the manufacturing context.

Academic and industrial research by Boehm [11] has shown that 86% of the total software development costs excluding maintenance costs are contained within the implementation and testing phases of the overall software life cycle which is presented in Figure 1.1. As these phases constitute the majority of the cost and resource expenditures for software development, the SPD methodology specifically addresses these life cycle phases (shaded boxes in Figure 1.1).

Organization

The second chapter reviews the literature pertinent to this research. Chapter III discusses the development of the system which is proposed for software planning and development management. The concept of the kernel and template are introduced and shown to be applicable to software development activities. Chapter IV explores the design of the various managerial and technical templates as they are utilized in the SPD methodology. The overall methodology is shown and then each series of templates which applies to that part of the methodology is detailed. Chapter V illustrates the resource allocation/cost modelling process. The personnel

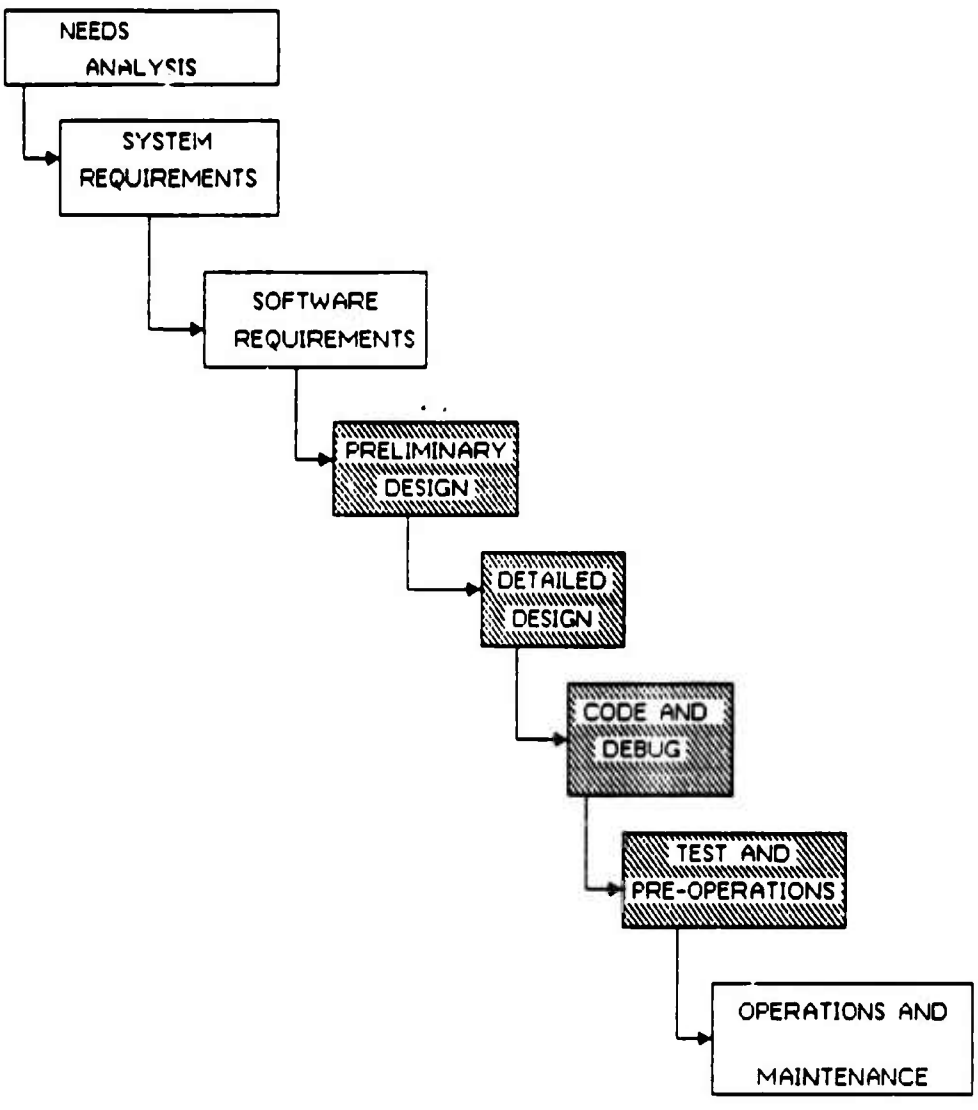


FIGURE 1.1--SOFTWARE LIFE CYCLE

resource is used as an example to show how the approach works. Chapter VI summarizes results of the work and makes recommendations for extensions.

CHAPTER II

PERTINENT LITERATURE

Overview of Recent Literature

Throughout the past ten years much work has been done to develop adequate cost modelling methods for software development. Putnam [70] developed the SLIM cost estimation model based on his analysis of the software life cycle in terms of the Rayleigh distribution of project personnel level versus time. Frelman and Park [26] report on the application of the RCA Price S model which is particularly developed for embedded systems in military applications. This model uses a two-parameter beta distribution rather than a Rayleigh curve to estimate the distribution of development effort versus calendar time.

Boehm [10] developed the Constructive Cost Model (COCOMO) for software cost estimation. The model is comprised of three increasingly sophisticated models analyzing work breakdown structure and phase sensitive multipliers for cost drivers. The basic model determines the number of man-months required for a particular software development from delivered source instructions. The total project development time is

determined from the estimated number of man-months. Equations for maintenance actions and tightly constrained embedded models are also included in the basic COCOMO model. The intermediate COCOMO model considers fifteen additional factors of software development. These factors include categories and subcategories of product attributes, computer attributes, personnel attributes, and project attributes. The COCOMO model uses multi-variate linear regression to determine the statistical coefficients required.

Thibodeau [86] proved that comparative results of software cost estimation models were possible. His research was inconclusive, however, as he was unable to obtain definite results because each dissimilar model examined was evaluated with different qualities of data subsets. No one standard data set was used to evaluate each of the different models. The best results were obtained using models with calibration coefficients against data sets with a minimal number of calioration points. Nelson [62] determined that of 169 different United States Air Force (USAF) software development projects, there were too many non-linear aspects of software development for a linear cost estimation model to adequately predict software costs.

Devanney [19] identified three possible factors that may contribute to software cost estimation errors:

1) Element of chance which makes cost estimation a random variable
2) The estimation technique itself
3) Non-uniform and unskilled application of the cost estimation technique

Wolverton [94] identifies traditional cost estimation techniques to include the following:

1) Top Down Estimating--This type of estimating relies on the total cost of large portions of previous projects that have been completed to estimate the cost of all or large portions of the project to be estimated. History coupled with informed opinion or intuition is used to allocate costs between packages.
2) Similarities and Differences Estimating--The estimator breaks down the jobs to be accomplished to a level of detail where the similarities to, and differences from, previous projects are most evident.
3) Ratio Estimating--The estimator relies on sensitivity coefficients or exchange ratios that are invariant (within limits) to the details of the design. The analyst estimates the size of a module by its number of object

instructions, classifies it by type, and evaluates its relative complexity.

4) Standards Estimating--The estimator relies on standards of performance that have been systematically developed. These standards then become stable reference points from which new tasks can be calibrated. This method is accurate only when the same operations have been performed repeatedly and good records are available.

5) Bottom-up Estimating--This is the technique most commonly used in estimating government research and development contracts. The total job is broken down into relatively small work packages and work units. The work breakdown is continued until it is reasonably clear what steps and talents are involved in doing each task.

The SPD methodology compares with these five different categories in a variety of ways. The SPD methodology is a top down hierarchical approach, investigating each layer of design sequentially. The use of the different types of kernels allows similarities and differences of software functions to be examined. The kernel/template concept allows a certain

standard to be designed for many types of different software application environments.

One clear conclusion that may be drawn from the GAO study is that some type of plan that encompasses both managerial and technical consideration is essential if the project is ever to deliver a product that can be used. In each statement of why the software was unacceptable to the government, lack of structured planning appears to be a main contributory factor.

Bender, et al., [7] reported that limited results were obtained for enterprise models and economic environment models, but that their use has been limited because of substantial development cost and difficulties in establishing the data relationships and interpreting model output. The term enterprise as used in this context means different types of companies or organizations. Thus, different enterprise models signify models developed for one particular type of company or organizational environment.

Research by Parker [64] found that most enterprises evaluated new software development projects by simply studying subenterprise or application areas and tnen attempting to integrate these various studies into overall strategic and operational plans. This approach was found not to be successful when the area was subject to a variety of external influences over which minimal

control could be exercised. What proved to be an optimal solution in one part of the enterprise was not necessarily overall optimal for the entire organization External influences are defined to be political persuasions, that is, increased requirements and add-on specifications after the initial needs analysis has already been accomplished.

Ahituv, et al., [1] found that lack of a documented project plan and an inadequately defined project scope were the reasons for a majority of project failures when vendors were used to construct software. Their research finds that a clearly written request for proposal (RFP) removes most of this type deficiencies between the customer and vendor. It was also determined by Cooper [17] that the successful software development project manager must have a software life cycle management plan that encompasses not only the software development phase, but all other life cycle phases of software as well.

Snyder and Cox [81] report that problems in software development occur because essentially static models of analysis and design might be avoided if more effort were expended in the analysis and preliminary design and development phases. Additionally, the benefits of greater emphasis on the developmental phases of analysis early in the system life cycle are

recommended by McKeen [53]. Parikh [63] states that the use of different software development methodologies can save substantial costs, depending on the methodology employed and the needs of the software.

Boehm [11] interviewed several senior executives of large companies and found that a viable strategic company plan and senior leadership commitment are the two vital parts of a successful software development program for any company.

Need for the SPD Methodology

The process of software cost estimation is very uncertain. No one cost estimation model or development methodology exists which is conclusively superior to any other. Most currently employed models use some form of linear regression to determine model coefficients. Research based upon a significant number of government software development projects concludes that a parametric linear cost relationship does not exist in software cost estimation, thus questioning the validity of the linear regression assumption.

Each of the above mentioned methods relies on either past work, that is, a job that was done previously and was similar (analogy), or some form of intuition on the part of the development analyst. None

of the methods discussed specifically divides resource categories into those of personnel, fiscal, and equipment within the estimation procedure.

Statistical distributions have been the basis for some cost estimation models. The Rayleigh distribution is used in the RCA Price S model. Other research, however, found that the element of chance makes cost estimation a random variable with non-uniform and unskilled application of the cost estimation technique.

The lack of an overall structured development plan provides various software designers no real way in which to uniformly manage any type of development action. In order to have effective project organization, both the management and technical elements must be simultaneously considered.

The SPD methodology presented here uses existing cost estimation models to help support resource requirements, but also provides the structure and focus of a basic design architecture. The existing cost estimation models are used before any actual resource allocation is performed. This allows the analyst to possess yet another input into the quantities of resources required in developing the software projects. This architecture uses the various types of templates and kernels to provide both technical and managerial structure to a development activity. The variety of

different templates allows different kernel types to be adapted to the specific development activity. The allocation approach considers resource subcategories and provides an estimation of the overall cost of the development activity.

CHAPTER III

DESCRIPTION OF SOFTWARE PLANNING AND DEVELOPMENT METHODOLOGY

This chapter introduces the SPD methodology and compares it to the software life cycle. Contrasts and differences between both are noted. The identification of salient technical and managerial elements required for software are identified and differences between requirements for in-house development and external contract vendor development are discussed. The construct of the kernel and template is defined and illustrated.

Scope of the Methodology

The correspondence of the classical software life cycle to the SPD methodology is presented in Figure 3.1. The software life cycle contains the needs analysis as an integral part of its overall system. This is not the case with the SPD methodology which requires that the systems specification be an input to the technical element. Since technical documents do not usually include the managerial resource needs and ramifications associated with technical specifications, managerial resource requirements are normally applied to

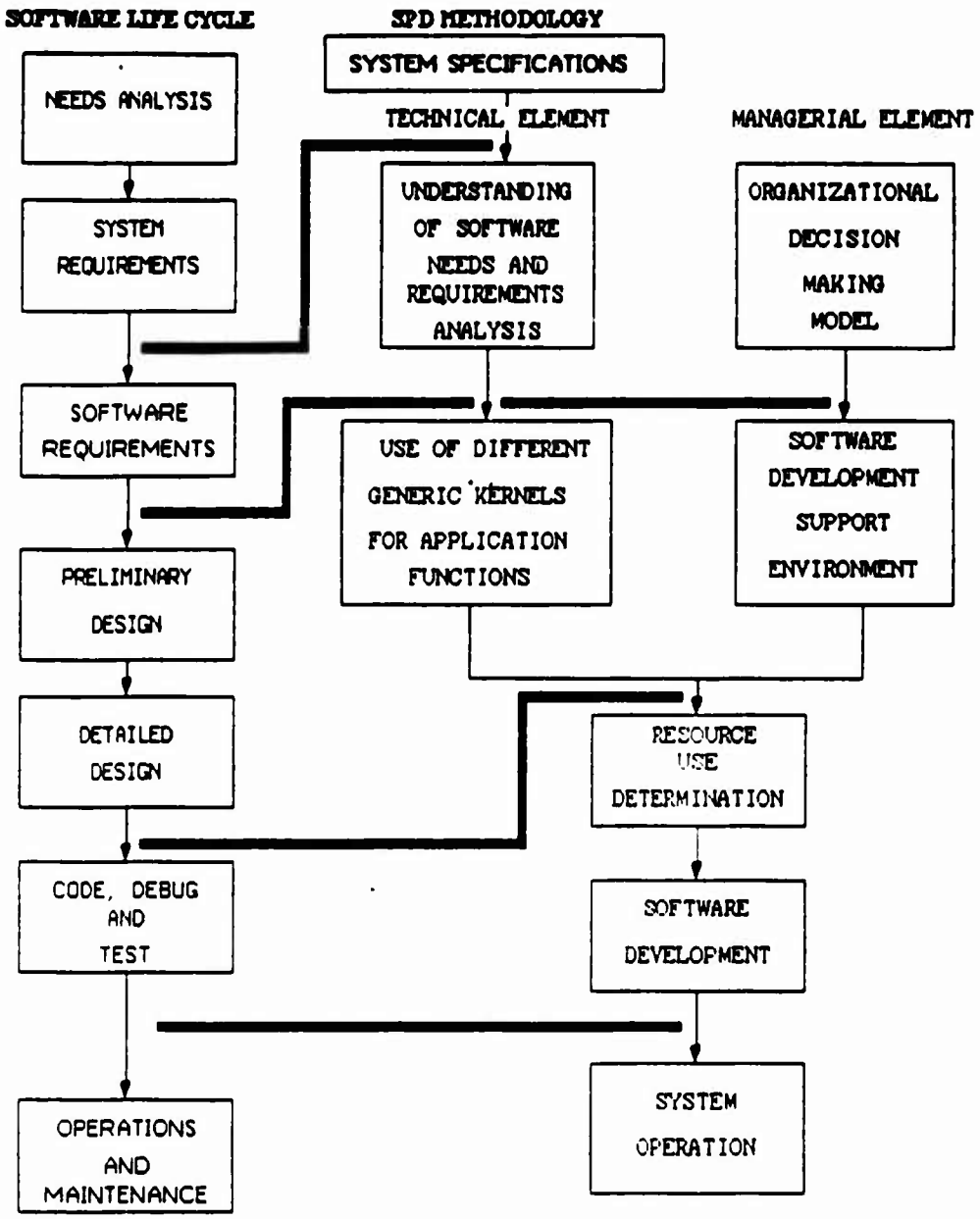


FIGURE 3.1--CORRESPONDENCE OF SOFTWARE LIFE CYCLE AND SPD METHODOLOGY

a project which is approved and deemed to be technically complete.

From the viewpoint of the SPD methodology, the understanding of the software needs and requirements analysis is analogous to the system and software requirements of the classical software life cycle. The preliminary design and detailed design steps of the software life cycle are analogous to specifying the different kernels for the various application functions. In the context of this research, a kernel is the nucleus or core of the functions that the software is designed to perform. The kernel will be discussed in greater detail later in this chapter.

The code, debug, and test blocks of the software life cycle are analogous to the resource use determination and software development blocks of the SPD methodology. The system operation block of the methodology compares with the operations and maintenance block of the software life cycle. Once the code has been developed, tested, and integrated, the SPD methodology stops since the planning and development phases are complete. The phase of software maintenance and enhancements/modifications are not specifically included in the SPD methodology.

This methodology introduces the managerial element in parallel with the technical element. Several

distinct benefits occur because of this alignment. Overall project communication between management, development staff, and the actual personnel associated with the results of the development effort is improved. Better knowledge and understanding of exactly what is required by the systems specifications provides for the allocation of the various resources required for the development effort.

Components of the SPD Methodology

The first block of the technical element of the SPD methodology (Figure 3.2) is defined as understanding of software needs and requirements analysis. The systems specifications document has stated this same requirement, however, the reason for inclusion of this block in the SPD methodology is that it provides a further clarification of what is required in terms of software systems and their interfaces with already implemented systems and databases. Many large software projects are developed which result in an end product that has failed to satisfy the original systems specification document. In some cases, another development effort is required to complete the design and implementation of software to fulfill the original specifications. The understanding block develops a

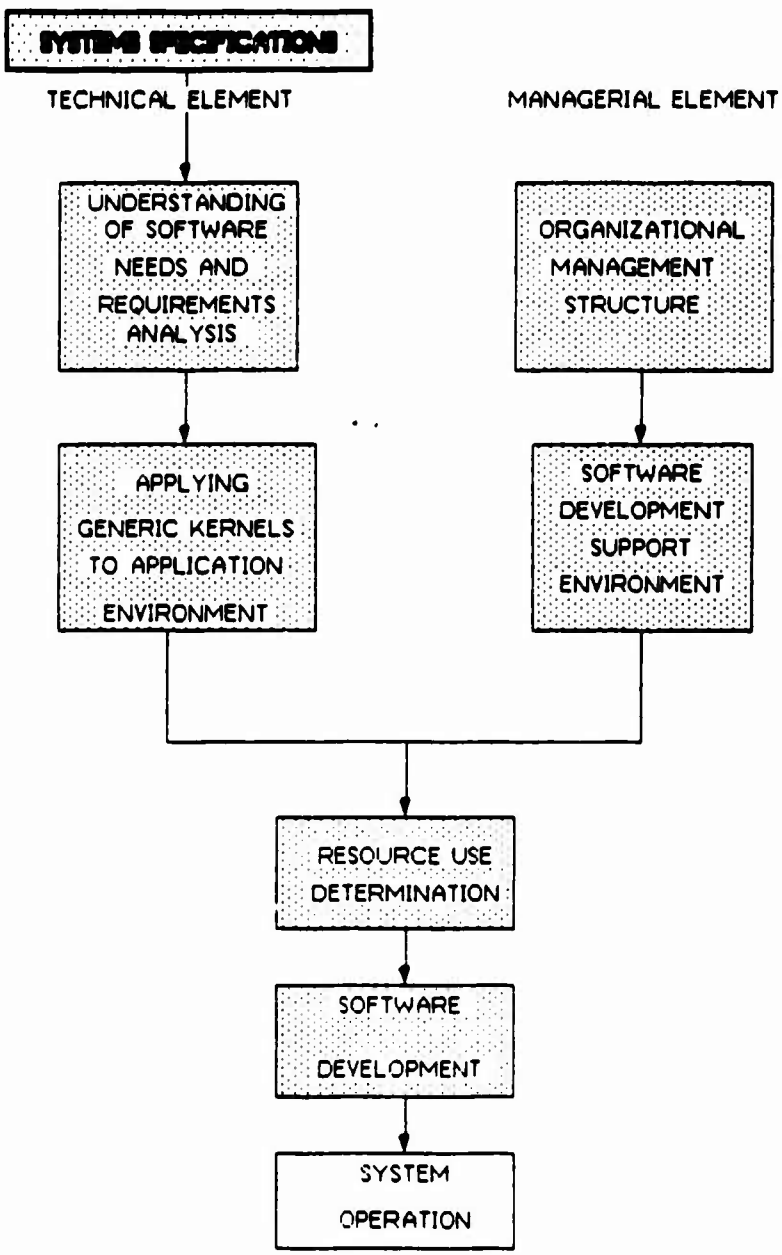


FIGURE 3.2--SOFTWARE PLANNING AND DEVELOPMENT METHODOLOGY

mechanism for increased communication between the technical and managerial staffs of a large corporate enterprise by insuring that both staffs fully understand what the software should do. If a disagreement or misunderstanding occurs, the requirements can be better defined and described before any substantial sum of fiscal and personnel resources have been expended.

Once an understanding of the software needs and requirements analysis is complete the number and role of the generic kernels is established. Generic kernels are used to determine the kinds and types of software required for application functions. The various types of protocols, interconnections, and data elements that are required for successful information integration are graphically depicted in the generic kernel.

The managerial element of the methodology is comprised of two blocks. The organizational management structure block is comprised of defining the overall managerial support for the project and determining the actual managerial structure of the software development team. It is suggested that an automation steering team be developed at senior management levels. This team would be comprised of functional representatives of the various departments affected by the introduction of new software. Representatives from finance, personnel,

budget, engineering, and supervisory production personnel should serve on the steering committee along with the director of software development. The director of software development would have a subordinate team whose sole function is to develop the required software specified by the systems specification document. This approach improves communication and cooperation between all concerned parties.

The software development support environment block requires the definition of the necessary tools and resources required to develop the actual software. Three components comprise this environment:

1) Hardware

2) Software

3) People

The hardware component includes the necessary types of machines required for the development and testing of the newly developed software. This includes equipment such as personal computers, executive workstations, and micro-mini and mainframe computers. The software component includes software development productivity aids and support tools which assist the programmer in developing the software. The people component of the support environment includes the necessary managers, engineers, technical support personnel, and programmers.

The resource use determination block is the key to

effective utilization of resources in the completion of a software development project. Within this block are included the two primary methods of completing the project. One for in-house software development, the other external contractor development. In-house software development is concerned with having the necessary resources required to develop the complete system using the corporate enterprise's own staff. If insufficient in-house personnel are available, either partial contractor support or full contractor support will be required. Partial contractor support would be required to augment those in-house personnel available to be assigned to the project while full contractor support would be necessary if no in-house personnel were qualified or available.

If a project is unable to be accomplished with organic personnel, the necessary scope, size, and pitfalls of the development effort are usually recognized after determining available resources. The scope of the overall project requires that the in-house staff be familiar with overall resource requirements. Many large software development projects have been completed by contractors with in-house personnel monitoring the contract. In most cases, when the contract changes are requested by the contractor they are granted because in-house personnel responsible for

contract monitoring fail to realize the overall scope and magnitude of the development effort. This methodology assists the in-house staff in understanding the overall scope of the software development project and determining the necessary level of resource expenditures.

The last shaded block in Figure 3.2 is labelled software development. This block is concerned with the actual coding process, tests for functionality, modification of the developed code, and integration of the code into plant operations. If the code is developed by an external vendor, consideration is given to the purchase of off-the-shelf code, modification of off-the-shelf code, or development of new code to fulfill customer needs. Training and documentation are also addressed within this block.

Introduction and Definition of the Kernel Construct

The SPD methodology uses the concept of a generic kernel (Figure 3.3) to model various software activities, processes, and functions. Each major type of process control, utility, or function that is to be performed by a software package or system can be represented by a kernel. Each kernel has different levels associated with it, with each level performing

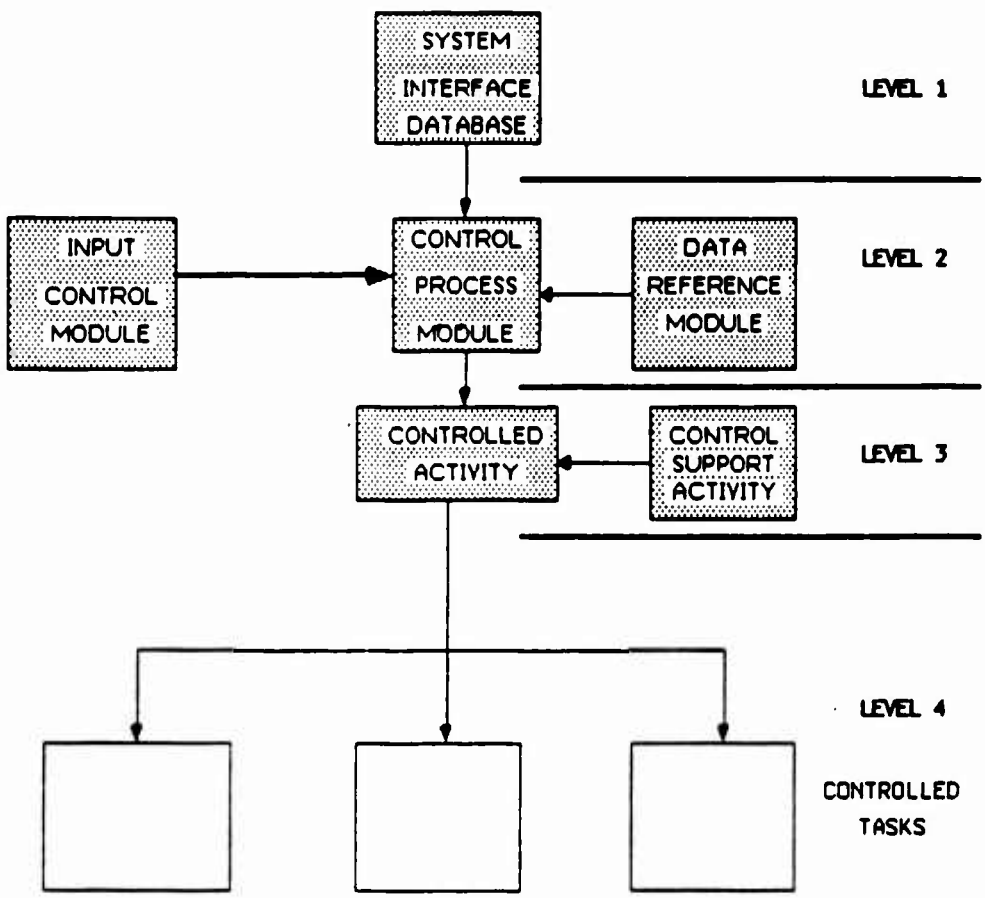


FIGURE 3.3--GENERIC KERNEL

different functions in the overall software package or system. Depending upon the type of kernels required for any one particular development, different levels could interface with various databases, have various communications protocols, application packages, utility programs, or database management and control systems to provide control to other parallel or subordinate kernel levels.

A certain hierarchy of ordered elements is inherent to the generic kernel. Level one components provide an interface to the organizational information system. Databases and programs at this level contain data about a particular department and certain utilities to provide the information interface to other departments within the corporate enterprise. The linking of two or more kernels is made at level one when more than one kernel is used within a specific application.

The number of kernels that are contained in any given software development project should be minimized. This minimization of different kernel types allows a integrated, homogeneous development while providing for kernels which contain a maximum number of similar functional elements, thereby improving overall coding speed and operational efficiency.

Level two modules provide the actual control of the activity that is being automated. The key module of the

kernel is in level two--the control process module. This module is integrally connected with the system interface database, and has as inputs the data reference module(s) and input control(s). The input control module allows for one or more different control actions that can be accomplished either in series or parallel depending upon the actual process. The data reference module can be attached to the control process module in either a serial or parallel configuration.

Level three modules serve as a buffer between the control process module and the actual controlled tasks that are found in level four. A buffer as used in the kernel context provides for connection of different code protocols, languages, or data types. It is possible that different language types and software tools already developed would be simultaneously contained within level three. A controlled activity could be a type of process, operation, or event that requires simultaneous inputs and outputs. The control support activity is designed to serve as a reference function for the controlled activity. This reference function could serve as a database, data element dictionary, measurement standard, or utility package required by the controlled activity.

Level four activities are the controlled tasks that receive driving signals from the level three modules.

The actual type of tasks may vary dependent upon organizational environment. However, these activities are at the lowest level in the organizational hierarchy. These types of activities include machines, purchase order writing, quality control functions, flow level control, vision inspection systems, and the like.

Each of the modules within the kernel are designed to perform a specific function regardless of the operational environment. The kernel construction allows for the possible introduction of off-the-shelf software to fulfill any module function. Each level of the generic kernel is designed to consist of approximately 1000 delivered source instructions of code. The 1000 source instructions size of the kernel is considered by industry to be a realistic kernel size [22]. Although the SPD methodology does not specifically use the delivered source instructions metric for a measurement standard, most individuals who are conversant with software design can easily relate to such a measurement tool. This metric provides a rough guide to nominal kernel size.

Applying the Generic Kernel to an Application Environment

To apply the kernel construct to an operational environment an example from manufacturing production

scheduling is developed in Figure 3.4. The level one function in the production scheduling example is the manufacturing database which provides the information interface of the manufacturing function to the other sectors of the enterprise. Most data common to the manufacturing function is usually contained within this database. There are three serially ordered input controls in level two for this example. The engineering design database inputs to the computer aided design process information giving the general specifications and characteristics of the parts processed through manufacturing. Process planning uses the design from the computer aided design and develops the controls for the production scheduling procedure. Data reference modules in this example are parallel in nature. Material requirements planning, automated storage, and retrieval and many other activities must be considered independently before the final production schedule is complete.

Level three activities include (at a minimum) computer driven and conventional machinery and the quality control module. These modules are equivalent to the controlled activity and the control support activity, respectively, in the generic kernel. The quality control module is labelled a 'controlled support activity' because of the supporting role that the

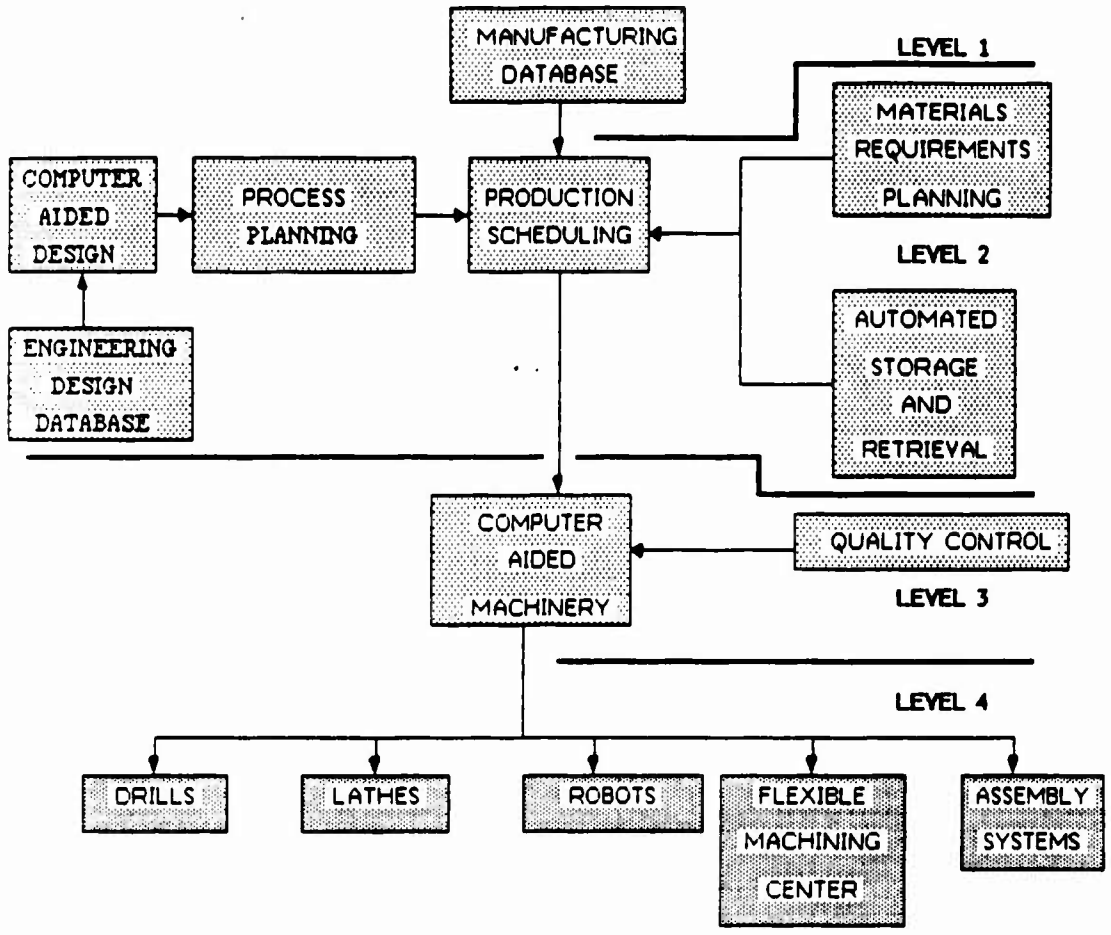


FIGURE 3.4--GENERIC KERNEL APPLIED TO PRODUCTION SCHEDULING

quality control function performs. Once the production schedule is developed, the various data is delivered to manufacturing system machinery so that the respective part can be produced.

Level four activities are the discrete controlled tasks which comprise the entire manufacturing process. Each of the various machines are assumed to be computer driven.

Definition of the Template

The SPD methodology uses templates as critical design tools that allow the various generic kernels to be tailored to a particular application environment. As discussed earlier, the generic kernel is applicable to a wide variety of software development environments. Thus, any software development project is initially comprised of generic templates depicting the various functions required of the software. Several of these templates are discussed in this section. Others may be added as the situation warrants. After the various templates are applied to the application environment, the kernel becomes tailored to the specific application environment under development.

The frequency in which the templates are used for any one particular development activity is contingent

upon many factors. Managerial thread templates would most likely have minimal change once the template was used. Once a team development managerial structure is designed, minimal changes usually result. However, this is not the case for the technical thread templates. Technical thread templates may be used several times in one development, depending upon number, complexity, and type of the various design kernels contained in the overall software development.

During the entire software planning and development process, the series of technical templates focus on the various technical considerations with which a software designer is concerned. These concerns are type of language (high or low level), protocols, interconnections (between different databases), machines, and types of data.

The managerial templates focus on matters such as software development team size, composition, and staffing. The software development team is actually responsible for developing, coding, testing, and implementing the software system. The types of various team members and the disciplines they represent, that is, engineers, accountants, supervisory shop floor personnel, and managers are also considered. The development team work environment is also a managerial concern. If the development process is to progress

smoothly, the different types of hardware available for the team to use, programming aids that enhance productivity, test bed environments for developed software are all managerial resources that must be planned and allocated.

Each template possesses certain characteristics which make it adaptable to many types of application environments and any size of software development effort. These characteristics include:

1) Use of generic template labels
2) Identification of interfaces, protocols, and data elements necessary for interconnections
3) Accommodation for a structured, hierarchical process necessary in software development.

Example of a Template

The example of a template (Figure 3.5) shows the various databases that are located within a specific division of an enterprise. The hierarchy of the databases are displayed, as are the different types of databases for the division. If this template were to be used in an actual development environment, the analyst would identify the applicable division and department level and lower databases, the relevant data elements in each database, communications protocols currently used

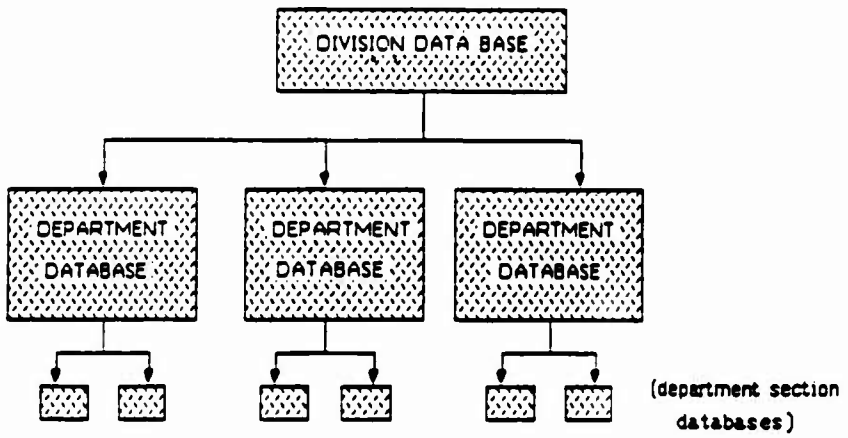


FIGURE 3.5--DATABASE TEMPLATE EXAMPLE

and those that may be required in the future to access the data, and the types of interconnections necessary. This identification of database parameters by this template allows a (generic) database kernel to be tailored to a specific application environment for database requirements. This same procedure would be utilized with each of the templates discussed later until each kernel is tailored to the selected application environment.

The various templates discussed later comprise the 'shell' of the actual software development. Each actual development project will be different and the shell is designed to accommodate those anomalies by aiding the analyst in an adaptation of the templates to the specific application environment for which the software is being developed.

CHAPTER IV

DESCRIPTION AND USE OF SPD TEMPLATES

This chapter divides the various SPD methodology templates into the three categories to which they apply. Each template series--technical, managerial, and resource use is presented in coordination with the overall SPD methodology as described in Chapter III. A description of what each template means and how it is used in the context of a software development environment is presented.

Technical Element Templates

There are five different templates (Figure 4.1) that apply to the technical element of the SPD methodology. Each template will be discussed except for the application of the generic kernel since this was described in Chapter III.

Constraint Identification Template

The constraint identification template (Figure 4.2) is one of the first steps in the use of the SPD methodology. Many software development projects have been initiated and terminated after significant

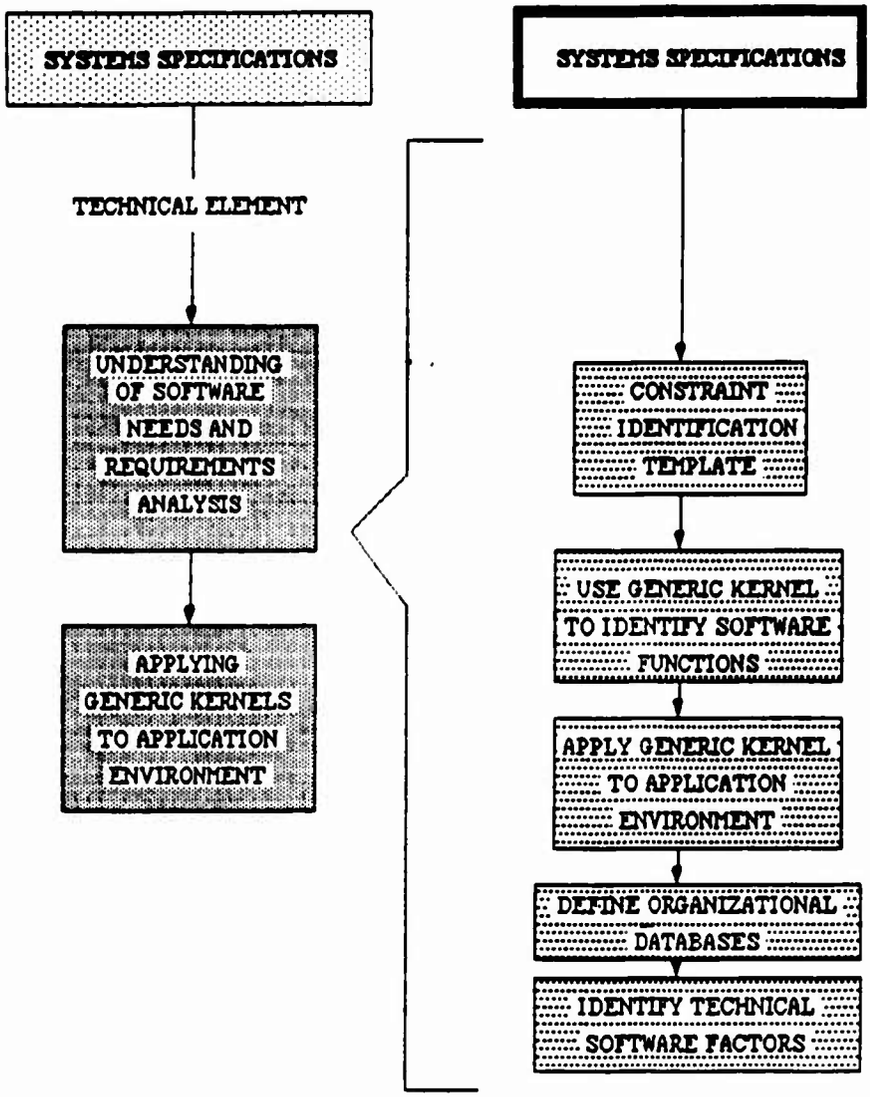


FIGURE 4.1--TEMPLATES WHICH APPLY TO TECHNICAL ELEMENTS OF SPD METHODOLOGY

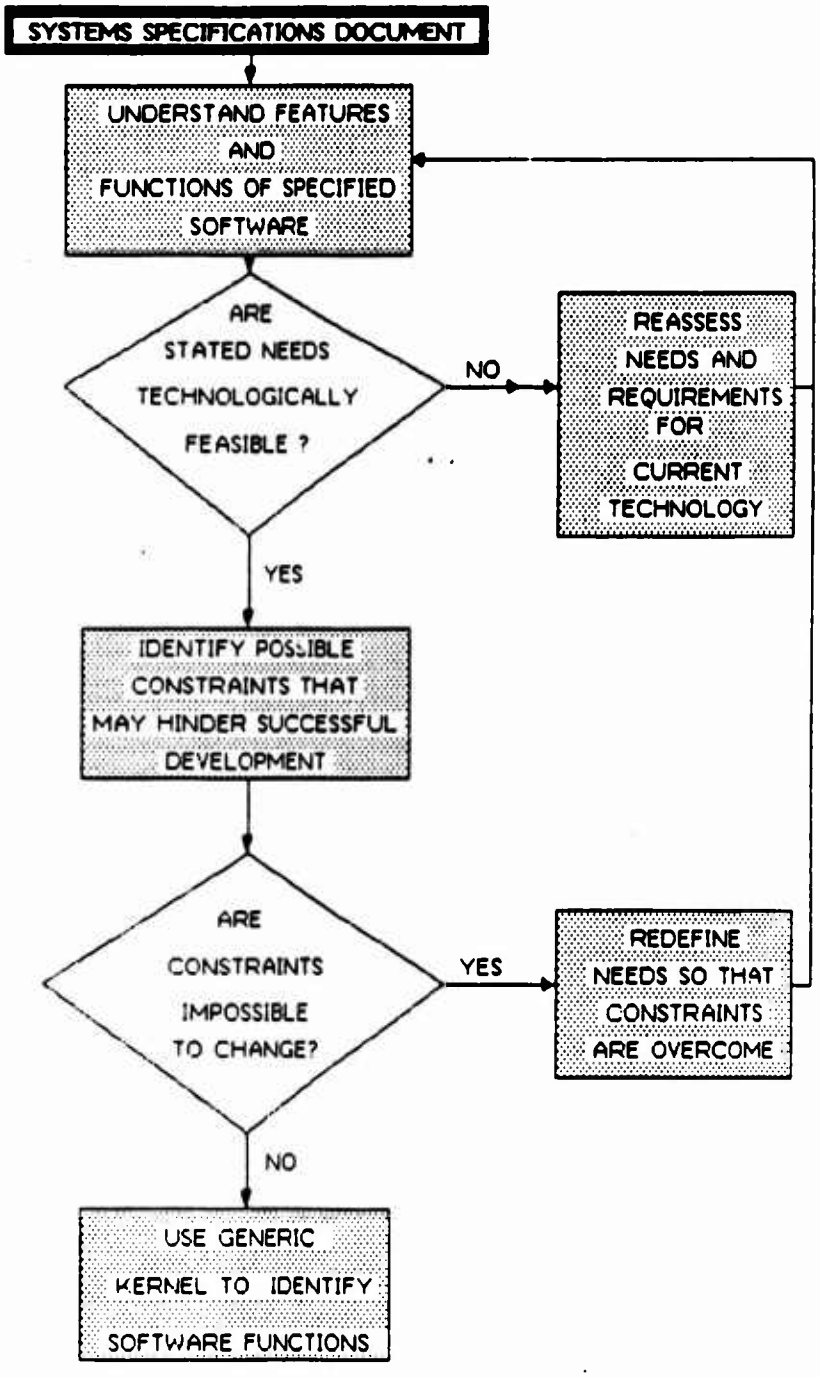


FIGURE 4.2--CONSTRAINT IDENTIFICATION TEMPLATE

personnel, fiscal, and equipment resources have been expended only to find that the requirements in the systems specification document were actually not technologically feasible or faced so many possible constraints that the project should have been declared infeasible at its inception. The use of this templa e allows the analyst the ability to identify possible technological deficiencies in a systems specification before an inordinate amount of work is expended trying to develop some system, process, or procedure that is not able to be successfully accomplished.

The systems specification document is an input to the methodology. One subset of the specifications document should be the software requirements which define the functions of all needed software and, therefore, the actual code requirements. The desirable properties of a specification should include completeness, consistency, comprehensibility, traceability to the original requirements, unambiguity (hence testability and/or verifiability), modifiability, and write-ability [92]. Costs and constraints should also be stated in the specifications document [74]. Knowledge of the final cost that the project is not to exceed as set by management and of the constraints that may impact the final product allow creative solutions to be investigated early in project design.

Graphic tools are available that assist the design engineer in software specification development. The use of the visual table of contents (VTOC) [41] provides a broad overview of the "modules" or paragraphs of the different source code requirements comprising a program. The VTOC provides a means for others to gain a broad scale perspective of the finished program and indicate how the corresponding modules are interrelated.

To meet the written software requirements using the SPD methodology the first step is to establish a complete and thorough understanding of the desired functions and features of the software. This step is one of basic engineering feasibility, that is, a determination of what is requested and if it can indeed be accomplished with available technology. The results of a state-of-the-art survey are used to determine if the software requirements represent a system that is technically feasible or if technology gaps are present which could prohibit the successful completion of the project. If the analysis finds the system as specified to be technically feasible, the next step is constraint identification (Figure 4.2). This step involves identification of any constraint that may sincerely hinder successful development. The key point is to remember that all major constraints that would hinder the implementation of the project should be identified

at this step. If the specifications are found to be both technically feasible and not severely constrained by factors that would hinder successful project completion, the generic kernels are developed to detail the software functions required to satisfy the specifications document.

If the systems specifications (or some part of them) are not technologically feasible, a reassessment of the specifications must be conducted for use with current technology. This is an iterative procedure that must be executed until the needs and current technology are congruent. The same logic applies to constraint identification and resolution. Satisfactory completion of both these steps is important before the determination and use of the generic kernels proceeds.

Generic Kernel

Once the technological feasibility and constraints are resolved, the use of the generic kernel identifies the various software functions. The description of the generic kernel and an application environment were presented in Chapter III.

Organizational Database Definition

The identification and definition of existing and required databases are important in the use of the SPD methodology. Knowledge of the location of various installed databases, the types of data elements contained within each database, the type of architectural structure of each database, and the required data communications protocols necessary between databases provide the system analyst with vital information. Knowledge of these above-mentioned factors insures that newly designed systems can be developed with minimal difficulty, cost, and redundancy.

Failure to spend adequate time in database definition before full scale systems development begins has proven to be extremely costly. General Motors studies [71] have found that approximately 50% of the integration cost is spent on communication equipment and 33% of the integration time and cost is expended developing custom software for each different type of device. These costs could probably be substantially reduced if the analyst investigated the database factors before beginning actual development.

Database definition should always be done for projects involving two or more databases. If only one database is involved, this step may or may not be

accomplished depending upon the level or depth of change being made in the respective database.

Many different databases may exist in any large corporate enterprise. The purpose of the template in Figure 4.3 is to identify each of the applicable databases at division level or higher in an organization. A division is comprised of a grouping of like-type activities or departments in the enterprise. Although the labels of the databases in Figure 4.3 reflect a manufacturing concern, note that many divergent databases are present in an organization. The personnel database may contain administrative data on the various employees, but also contains skills inventory and work standards necessary for different types of manufacturing activities. The financial database may contain budgetary and financial information on the overall economic condition of the enterprise. The marketing database may contain client lists, new orders, descriptions of the various products that are manufactured, and other types of sales information. The engineering database may contain engineering drawings, engineering change requests, and currently developed products that are being engineered but not yet in production. The manufacturing database contains master schedules, work in progress, work completed, parts on order, and inventory. Certain manufacturing standards

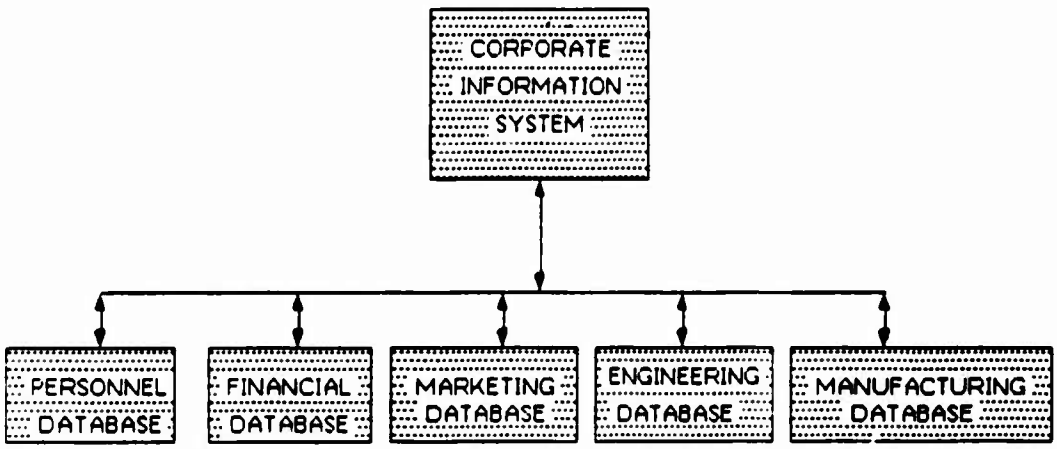


FIGURE 4.3--TYPICAL ORGANIZATIONAL DATABASES

including time, labor, or parts consumption (bill of materials) may also be included in this database.

After the divisional databases are identified, the next step is to identify departmental databases. Figure 4.4 identifies a database hierarchy in which a division database has three subordinate departmental databases. In each level, the identification of data commonalities, data elements, communications protocols, and unique computer requirements must be completed as explained above.

This same hierarchical process of database identification must be completed to the lowest level database in a department. Figure 4.5 illustrates a possible divisional database hierarchical schema in which departmental and lower databases are classified according to their application, not their resident organization, that is, department. In some cases, the task unique databases may be difficult to locate or non-existent. Some of the lower level databases may have information contained in a variety of locations or on several computer or manual systems, that is, not in the form of a typical computer database. Often these files are as important as the data contained within large, formal, and maintained databases.

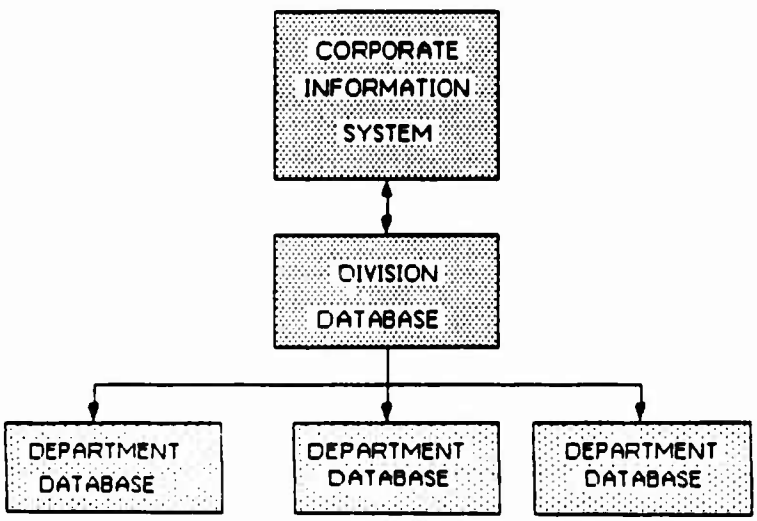


FIGURE 4.4--DATABASE HIERARCHY

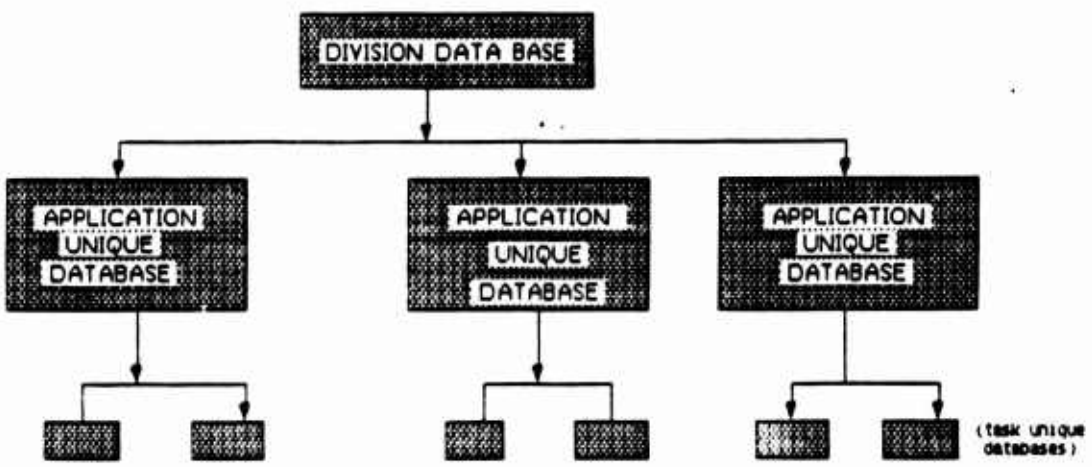


FIGURE 4.5--DIVISION HIERARCHICAL DATABASES

Software Technical Factors

There are many different technical factors that comprise software development. However, five of them (Figure 4.6) are considered significant when considering overall resource planning. Whenever a large software development project is to begin, substantial thought must be given to the initial functions to be considered for automation. In the context of this research, a function is defined to be a set of related operational activities that perform a distinct action or control upon some type of work place.

Most development engineers have found that introducing automation oriented software incrementally into the plant is more satisfactory than to attempt automation of the entire plant at one time [37]. A systematic, incremental introduction of automation allows learning and problem resolution before significant overall difficulties occur. In a manufacturing context, consider as prime candidates for automation those functions where the smallest amount of recognizable change or least possible factory line disruption will occur. Such functions usually provide the least disruption to the overall factory process and may be placed on the previous operation method if software problem resolution is required.

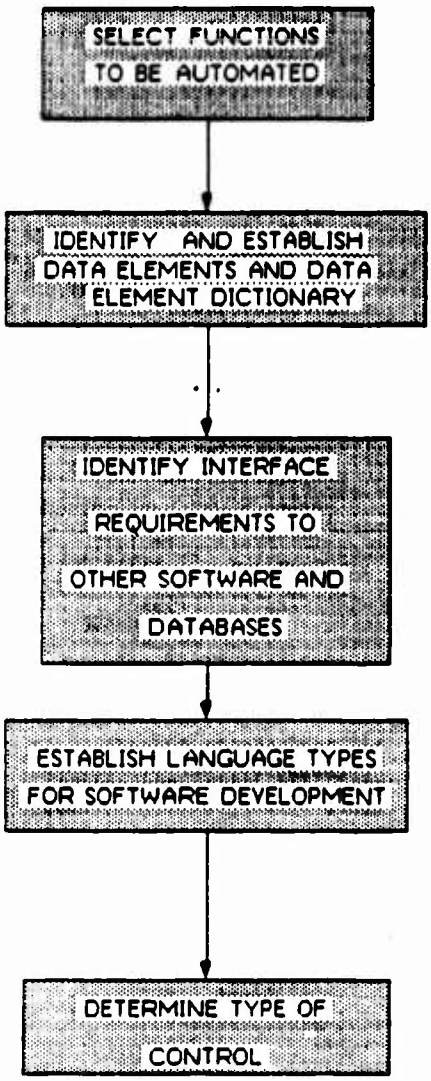


FIGURE 4.6--SOFTWARE TECHNICAL FACTORS

If an enterprise has had previous experience with automation software and information systems integration, the identification of initial functions is required to recall the problems encountered during previous automation projects and to insure that those problems are not repeated. This may be done through a meeting of those involved in the current software development activity and the personnel (if possible) who had responsibility for the previous project.

After the functions for automation have been identified, new data categories and the addition/establishment of the data dictionary must be considered. This step is placed after the identification of functions for automation for several reasons. Knowing which functions are initially automated and the data categories involved helps the analyst identify the need in the software development effort. Also, knowledge of the data categories involved in the automation project assists in the identification of the various interfaces and data exchanges that may be required.

Often individual software projects are completed and found to be independently functional, however, once an attempt is made to integrate the discrete package with a system, the problems are identified. Identification of the data interfaces provides

knowledge of the various protocol requirements and should prevent substantial interface problems.

The language in which the software will be developed is selected after the successful identification of the data categories and interface requirements. Higher order language possess distinct benefits for software development, especially in terms of productivity of programmers. Documentation and maintenance of a higher order programming language is considered to be easier than working with a lower level language [79]. Maintenance and problem resolution on lower level languages usually requires more time to debug because of the amount of time necessary to learn exactly what the program is doing [27]. Language type is a major concern of the development engineer as personnel resources are a large factor in the cost of overall software development. Detailed discussions of different language choices is presented in [55] and [77]. The C, LISP, and PASCAL languages are discussed in [16].

The determination of the type of control which the software will provide is the final technical factor that requires consideration. In most industrial automation applications real time control is necessary. Batch control is seldom preferred because of the dynamic characteristics of the manufacturing environment. Often

automation projects developed by programmers who are highly experienced in batch mode development and naturally feel qualified to develop real time software, have poor results. This experience may be present because programmers who do not possess a basic familiarity with real time control software may be unsuited to develop automation software which has a real time control. This unsuitability is caused because of the dynamic, interactive characteristics of real time software when compared to software developed in a batch mode. Real time software is dynamic in nature, that is, parameters and data elements are constantly changing, depending upon the application. Programmers who fail to possess these characteristics and the necessary 'mindset' that is required for real time control attempt to force the system design into a batch mode orientation.

Each of the technical factors described above are important in determining the overall resource expenditure for automation software development. Failure by the software development engineer to consider any one of these factors when determining the scope of the development effort may possibly result in an underestimate of the several resources required.

Mangement Element Templates

There are four different templates that apply to the management element of the SPD methodology. These templates are identified in Figure 4.7 and described below.

Managerial Decision Making

In a corporate enterprise decision making usually occurs at three levels. The strategic level is concerned with long range planning, competitive position, market share analysis, research and development, and financial posture for the planning horizon of five to fifteen years [38]. This level is most concerned with concepts, policies, and how new innovations and products may substantially contribute to the overall competitive posture of the enterprise.

Contrasted with the strategic management level is the tactical level which focuses on decision making within the enterprise itself. Concerns at this level utilize a planning horizon of one to three years and primarily address production quotas, assigned goal attainment, and implementation of policies made at the strategic level [42]. Less conceptual planning is done at the tactical level since more attention focuses on achievement of enterprise goals.

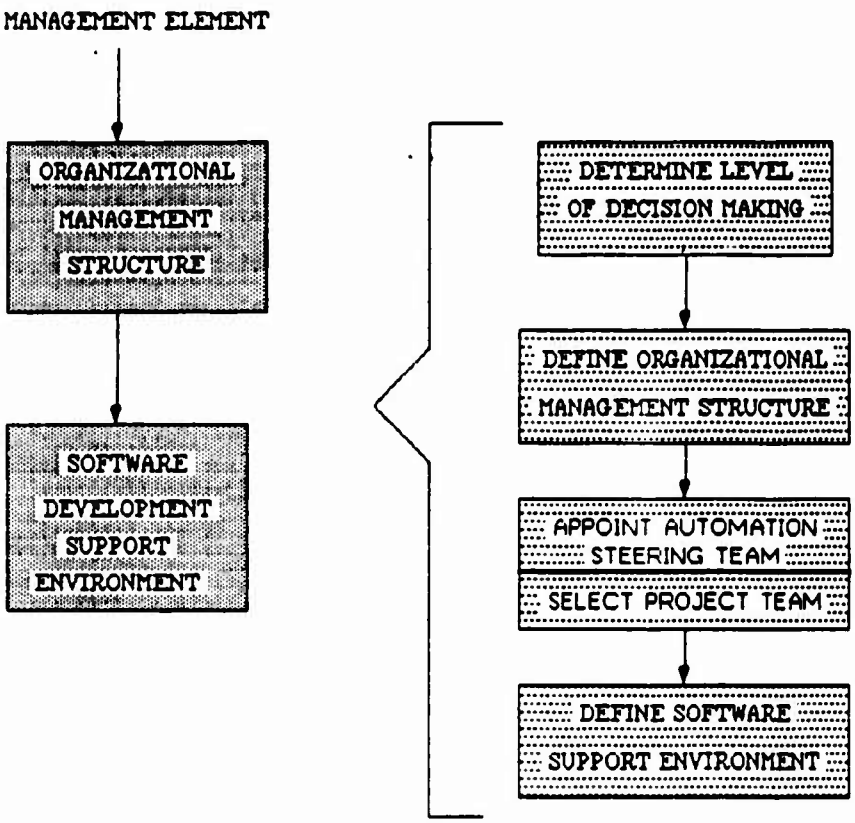


FIGURE 4.7--TEMPLATES WHICH APPLY TO MANAGERIAL ELEMENT OF SPD METHODOLOGY

The operational management level is at the heart of corporate enterprise and is responsible for planning and execution of the tasks that make the enterprise operate. The planning horizon of decision making is very near term, usually several months up to a year [31].

Senior level management support is necessary for a successful large scale software development project. This support should take the form of dedicated personnel, equipment, and fiscal resources required to accomplish the task, and more importantly, a will and dedication to implement new and significantly different operations procedures once the project is complete. This type of support is garnered from a senior executive who has a vested interest in the success of the particular system and who has overall responsibility for primary user departments [30]. This individual should be at the vice president level and have a significant influence in overall corporate decision making.

Organizational Management Structure

The organization chart in Figure 4.8 is adapted from Koontz and O'Donnel [43] and shows a typical manufacturing organization. The company has four divisions which are headed by vice presidents. Numerous

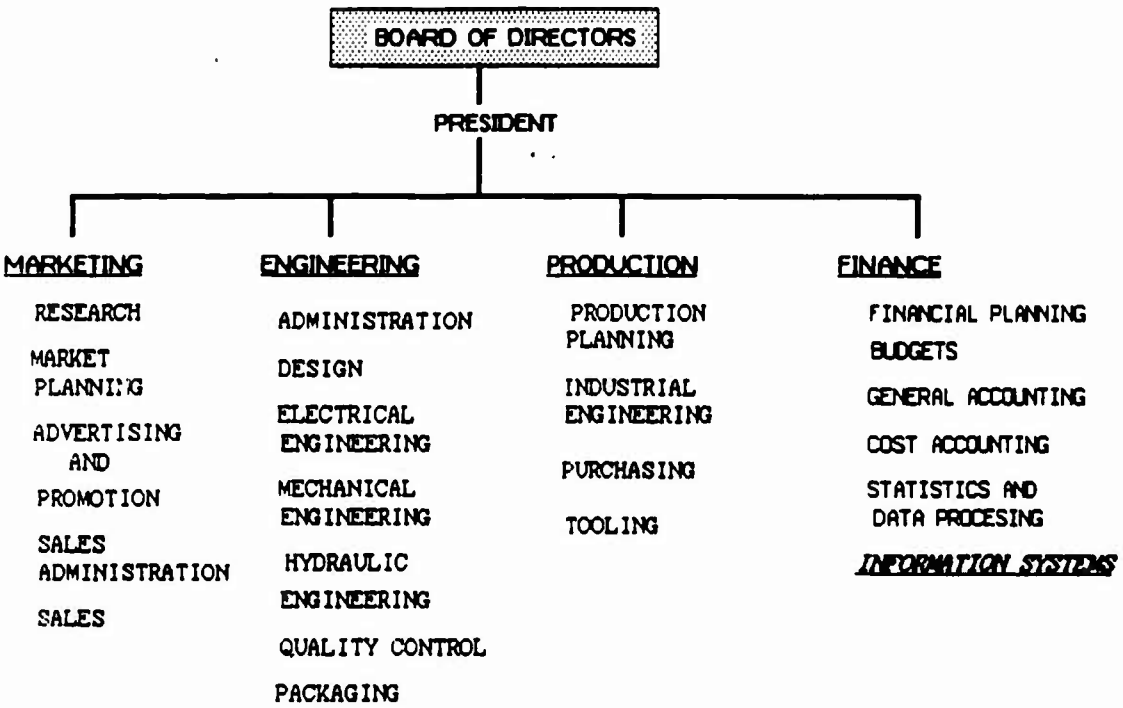


FIGURE 4.8--CORPORATE ORGANIZATIONAL STRUCTURE

subordinate departments are contained within each division. Although the vice presidents of engineering, production, or finance might be equally well suited as the automation sponsor, in this case the vice president for finance is probably best suited for the task because this division is responsible for financial and economic planning decisions. More important, however, is the vast experience that this individual possesses with different types of information and corporate data automation systems. The broad experience gained from knowledge and use of other corporate data systems allows the financial vice president to effectively articulate to senior corporate management the benefits and results obtained from automation software.

Automation Steering Committee

The vice president for finance may be the senior executive responsible for automation and software development, but the specific managerial, technical, and administrative details are best left to an automation steering committee (Figure 4.9). This committee is structured to function as a working committee to implement policy decisions. The committee should report to the top echelon of the organization and have delegated to it specific discretionary authority. Each

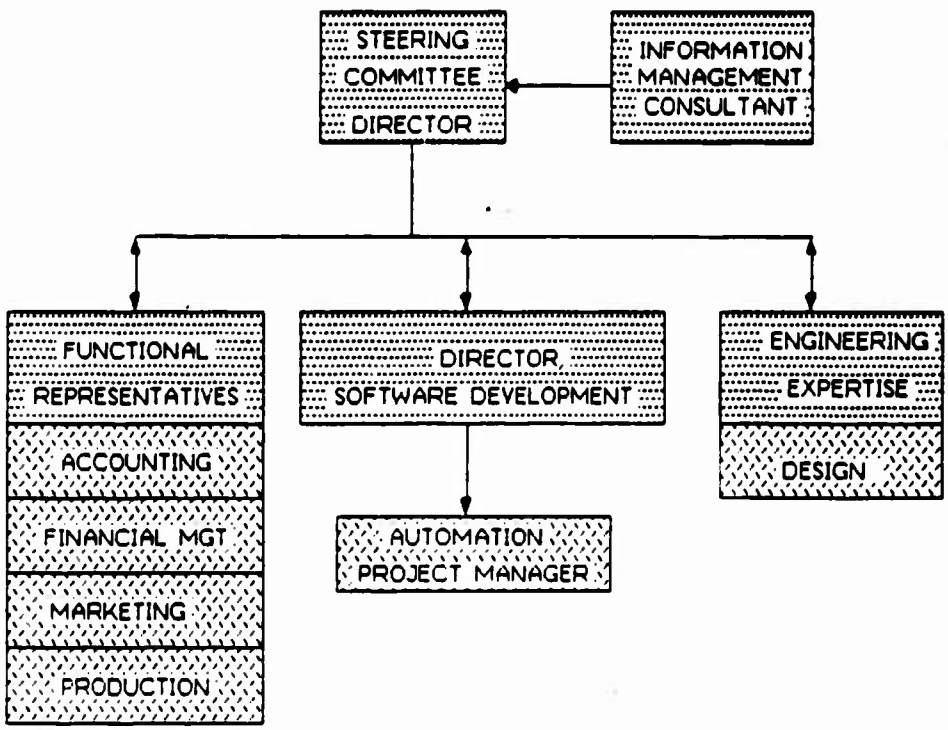


FIGURE 4.9--AUTOMATION STEERING COMMITTEE

steering committee member is partially responsible for the effective use of the resource the committee oversees. The committee has the power to establish priorities, control expenses, make economic and policy rulings, and provide policy direction to the automation project manager [88]. The director of the automation steering committee should come from the information systems department under the vice president for finance. This individual will be responsible for the day to day activities and management of the steering committee.

Because automation software development crosses many functional lines, various users should also participate in the steering committee. These users should represent each of the activities that may be influenced or effected by the software development activity. Thus, representatives from finance, accounting, production, and engineering are all potential steering committee members. The assignment of production personnel from the shop floor has been found to be very beneficial in obtaining labor personnel support for increased industrial automation [58]. A complimentary benefit of production personnel membership on the steering committee is that they can usually identify problems with certain functions to be automated before the actual development and implementation takes

place. The production personnel should be considered and respected as "functional experts" in their particular area.

An information management consultant may be used as a technical advisor to the steering committee director if the director fails to possess the requisite technical training necessary to successfully complete the automation project. The role of the advisor is to provide technical guidance about the various components and elements of the automation system [44]. This individual can be a commercial consultant, a university professor who possesses a background in factory automation, or an individual, who by virtue of previous work experience in industrial software, is qualified to address technical questions.

Other research has shown that user participation in software system development is effective if users exert influence toward both conflict generation and conflict resolution [73]. Many territorial conflicts exist in development of automated information management systems. User participation on the steering committee tends to reduce this type of conflict.

Further discussion of the benefits and viability of the steering committee is contained in [15, 21, 69].

Software Development Team

The structure and composition of the software development team can take on a variety of forms. Organizationally, software development teams can be divided into many different types [3, 6, 12, 54, 67, 68, 90]. Three distinct types of software development team organizational structures are functional, project, and matrix. The first accommodates the functions of programming, systems analysis, and computer operations. Each function has a manager and each manager reports to a department head [18]. Project teams are formed combining a mix of skills under a project leader who is responsible for accomplishing all the various tasks required for a given system [84]. Matrix organizational development concentrates on interdependence of functional elements and project elements. These are comprised of overlapping, task-oriented friendship groups of people with complimentary specializations [72].

One type of project development team is shown in Figure 4.10. The project manager is the same individual labeled automation project manager, who is subordinate to the director of software development in Figure 4.9. In this organizational structure task managers supervise two or more line managers who oversee the various types

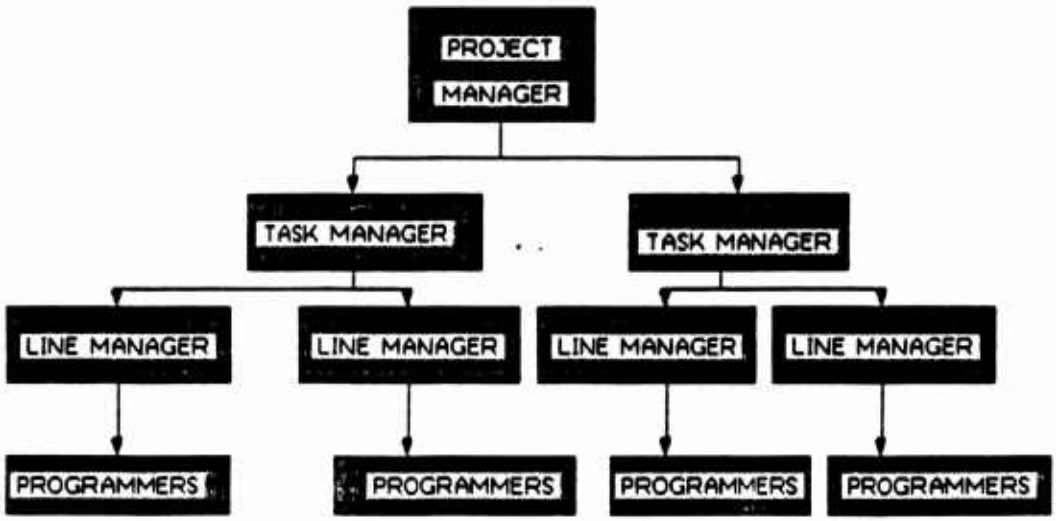


FIGURE 4.10--EXAMPLE OF PROJECT DEVELOPMENT TEAM STRUCTURE

of programmers. A task is defined to be major subsections of code which perform a certain software function. Depending upon the size of the task, this could include complete kernel development.

Software Development Support Environment

To maximize productivity from the software development team, a complimentary software development support environment must be created and maintained. A software development support environment consists of (1) techniques and automated tools which assist developer of software systems, and (2) an organizational structure to manage the process of software production [91]. Three key factors of the support environment depicted in Figure 4.11 are people, software, and hardware. The people component includes managers, programmers, and clerical personnel. The hardware component includes the development system, the target machine, and other hardware elements of the new system. The software component includes support and utility software necessary for the development of automation software.

The hardware component as shown in Figure 4.12 is composed of five types of hardware. The communications category includes modems, acoustical couplers, local area networks, and other communications devices required

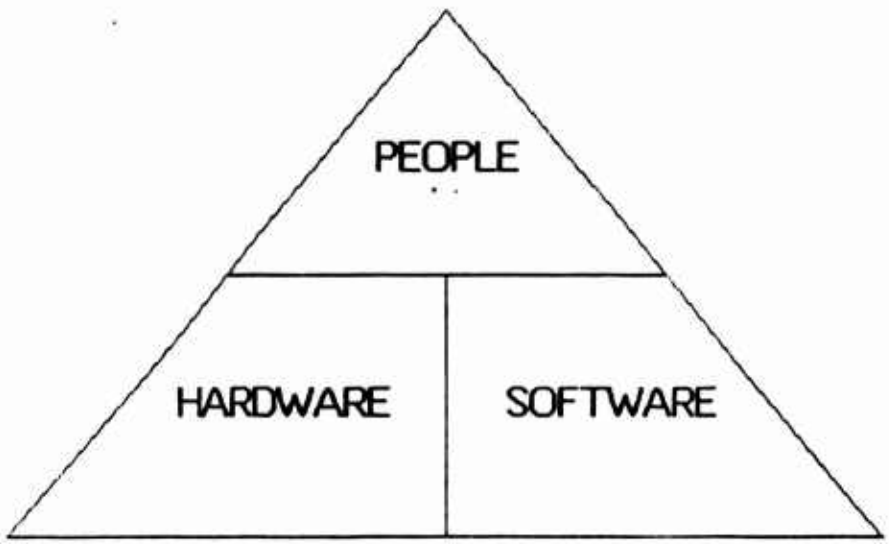


FIGURE 4.11--SOFTWARE DEVELOPMENT SUPPORT ENVIRONMENT

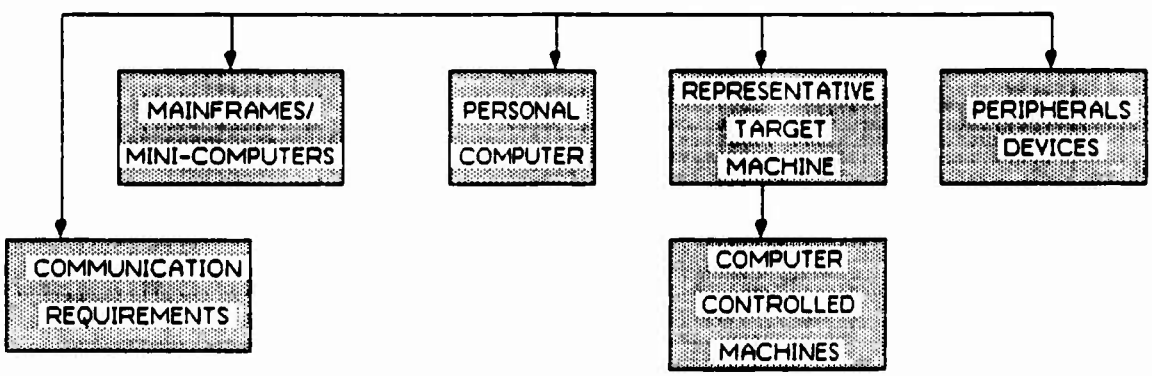


FIGURE 4.12--HARDWARE COMPONENT OF DEVELOPMENT ENVIRONMENT

to assist software integration. Mainframe/minicomputers are host computers that are available for the development team to use for the software project. In a large software development environment where many different types of software are being developed simultaneously, target (host) machines which operate the developed software are necessary. If the software is to be developed by a contractor, the contractor may have machines similar to those the actual developed software is designed for. Personal computers and executive workstations are hardware components of the development environment that can be used in either a stand alone mode or can be networked with mainframes or minicomputers.

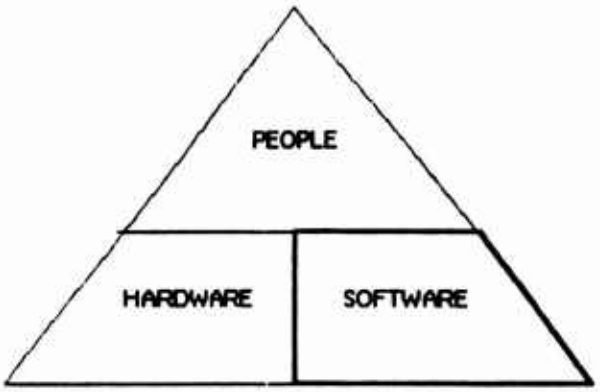
The representative target machine and computer controlled machines are devices on which the developed software will actually operate. Computer controlled machines can be either scale model equipment or the actual machines that will operate the developed system.

Peripheral devices are defined to include plotters, printers, display devices, and other hardware components that will be used with the developed software. Each of these devices are used to ensure that the software provides the designed responses necessary when the system is operational.

The software component of the development environment as shown in Figure 4.13 is comprised of five different resources. Each of the resources is used to not only increase the actual productivity of the software development team, but aid in the actual development and testing of the software.

One necessary resource is structured software development tools that offer a programmer an architecture for the overall development effort and may enhance his/her productivity. Modular programming, process ordered design technique, single independent functions, and single exit/entry point modeling methods are some tools discussed by Yau and Tsai [95]. Functional decomposition, stepwise refinement, and other design techniques should also be considered [65, 93]. The structured design approach [57, 97] which maps the data flow of a problem into its software structure using design analysis techniques is useful. The Structured Analysis Design Technique (SADT) is a graphical language for explicitly expressing hierarchical and functional relationships among objects and activities [75, 76]. This system structure graphically highlights software interfaces which can be used in top-down, structured, modular, and hierarchical architectures.

Some data structure design methods which emphasize problem structure, construct architecture, and detailed



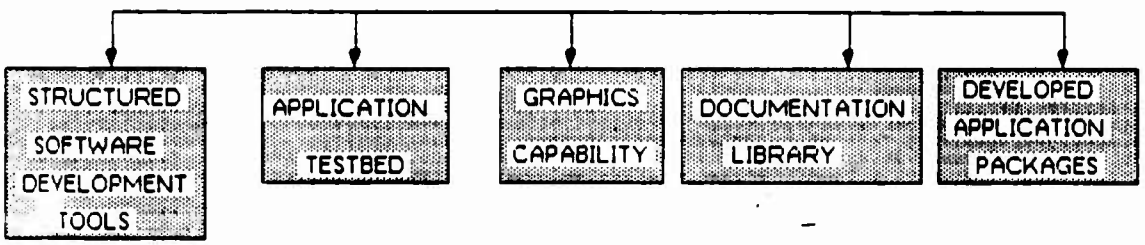


FIGURE 4.13--SOFTWARE COMPONENT OF DEVELOPMENT ENVIRONMENT

design concurrently are presented in [40, 92]. The Hierarchy, Input, Process, Output (HIPO) method provides the ability to represent the relationships between input/output data and the software process (66, 83). Nassi-Shneiderman diagrams [61] provide a well defined functional domain which simplifies the determination of scope for local and global data. A control free grammar structure is designed for use with structured programming and top down methods [14]. The method uses structured English and a syntax similar to that of a programming language. Additional structured software development tools are discussed in [13, 28, 29, 30, 49, 50, 51, 52]. These methods allow the programmer wide latitude in the type of tool selected and method of utilization in an actual development environment.

The application test bed resource (Figure 4.13) is coupled with the equipment listed in the hardware component to allow the developed software to be tested as it will actually be employed in the operational environment. Testing is necessary to determine if errors exist and to identify discrepancies between a product and its specification. There are seven different categories of software tests [35]: demonstration, benchmark, complete feature test, new feature test, performance test, reliability test, and stability test. Each of these is explained in the

section entitled In-house Code Development and Integration.

A graphics capability is listed in the software component so that graphics can not only be used in the developed software, but to also capture graphics output produced by the software. This graphics capability is comprised of automated drawing aids, necessary computers, plotters, and driving devices required to allow full operation of software capabilities.

The documentation library contained in the software development support environment is a key tool for the programmer and manager [4]. Documentation must be the same quality as the developed software and adequate enough to fully explain all types of software functions [56]. Documents contained in the library include the following [20]:

Problem definition documents

System, module, and component specifications

Programs and their descriptions

Test specifications, test drivers, and test data

An additional resource in the software component of the development library are preprogrammed application packages. These packages are designed to fulfill a variety of needs commonly encountered with software development. Networking, file handling, graphics,

programming language compilers, and communications packages are just some of the various packages that could be available.

The people component of the development environment (Figure 4.14) involves the actual personnel that will be involved with developing the software. The automation project manager, who is responsible for actual code development, is usually an individual who has an excellent background in computer science or engineering and who clearly understands not only the management of large scale software development, but also the various technical implications of different types of coding methods.

Test development engineers support the automation project manager in the conduct of the various tests. These individuals are responsible for the test plan development and test execution to see if the developed software performs as described in the specifications document. The individuals who develop the test plan are independent of the programming staff. This independence ensures test objectivity and early identification of any flaws or problems in the developed software.

The support staff includes librarians, documentation clerks, and technical illustrators. The librarian is the interface between the programmer and the computer. All changes in the project library are

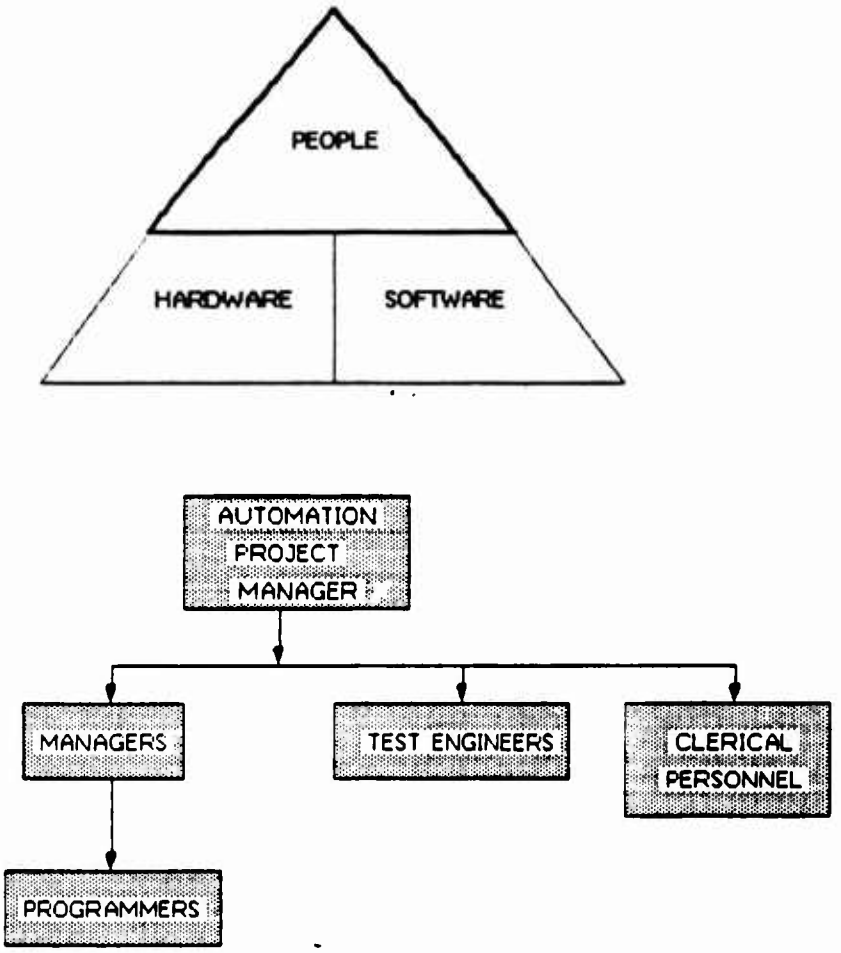


FIGURE 4.14--PEOPLE COMPONENT OF SOFTWARE DEVELOPMENT ENVIRONMENT

handled by the librarian [98]. In most automation software environments, self documenting software is used to reduce the burden on programmers and to insure tne conciseness of the documentation while the development effort proceeds. Problem statement languages and analyzers allow system development activities to be recorded using a computer database to store all basic systems data [85]. Some clerical personnel are still needed, however, to maintain the other paper products previously identified as being stored in the library.

At this point all of the templates that comprise the technical and managerial elements of the SPD design methodology have been explained. If a new software development project is just beginning, each of the templates would be used to tailor the generic kernels to the specific application environment. If a modification or enhancement to an existing system is the main thrust of the software development effort, it is possible that not all of the templates would be required because of the existing organizational structure, available data and databases, and presently operating systems. However, each template should be consciously considered before eliminating its use to ensure a completeness of planning.

Resource Use Determination

After the technical and managerial phases of the methodology have been performed, the next step is to determine the various resources necessary to implement and complete the software development project. The four templates shown in Figure 4.15 address the resource use determination phase of the SPD methodology.

Resource use determination is placed at this point in the SPD methodology (after the managerial structure has been determined and the various technical questions have been resolved) because it allows the user to have a good understanding of what is required for the software project. If in-house resources are not available and contract vendor resources must be employed, the in-house staff has a good knowledge of the magnitude of the overall project to be accomplished by the vendor.

Determination of In-House or Vendor Development

The question of sources of software development is dual-faceted; either develop the required software in-house or contract the development effort to a qualified vendor. Several questions must be addressed before a clear determination can be made to use either method. Figure 4.16 is a template useful in

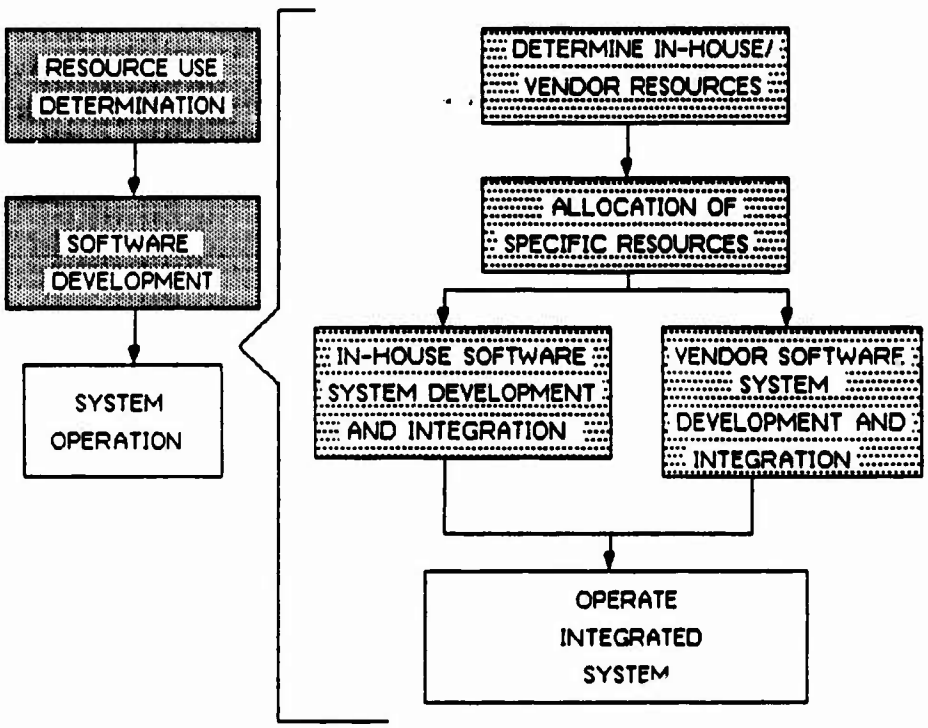


FIGURE 4.15--TEMPLATES WHICH APPLY TO RESOURCE USE PHASE OF SPD METHODOLOGY

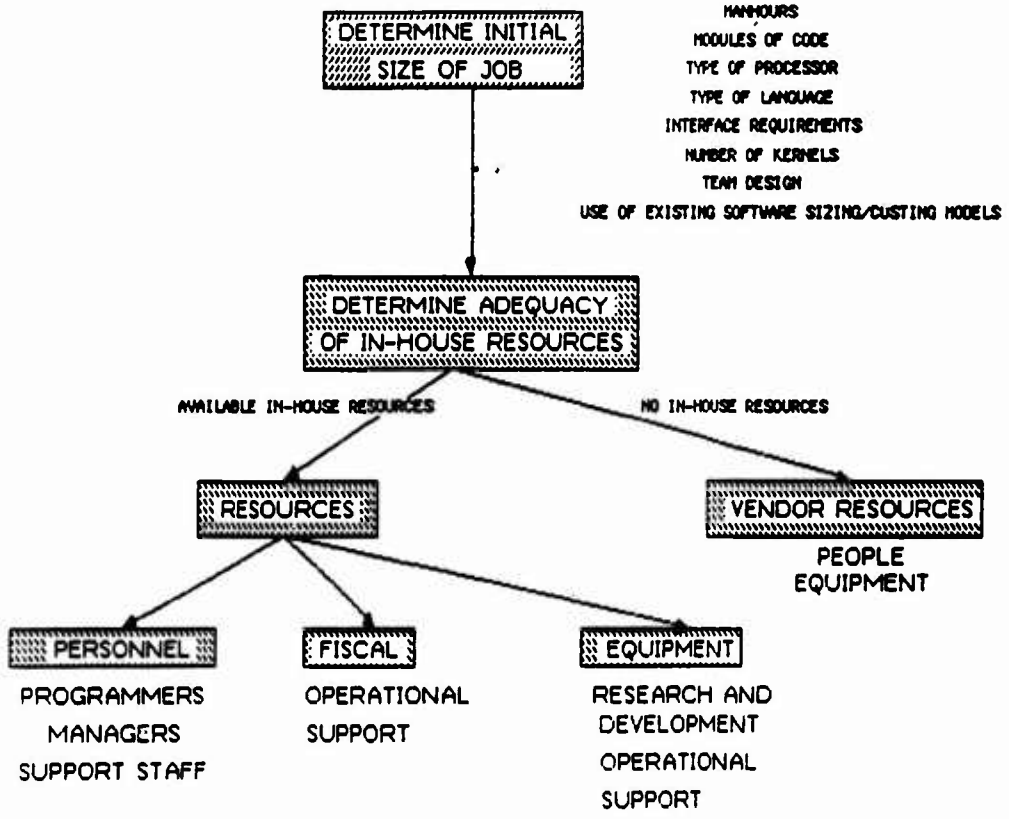


FIGURE 4.16--RESOURCE ALLOCATION TREE

ascertaining which is more judicious based upon the quantity of resources available.

The initial job size is the first step to be determined. There are several software engineering metrics listed beside the first block that can be used [5]. Because this methodology employs the concept of the kernel, determining the number of kernels required for the effort is an important metric. The complexity of some software development projects requires that the quantity of kernels, coupled with one or more of the listed metrics, may be required to ascertain an (the) initial size of the job.

Existing software size and cost estimation models can also be employed at this step. A variety of models exist, but few proven formulas are known [48]. Thus, each model uses different metrics and factors to determine size and resource requirements. The existing models are in no way intended to supplant the SPD methodology, but only provide additional information about overall size determination.

A variety of different software sizing and resource estimation models are discussed by Moranda [60]. Robust mathematical treatment of the Norden-Rayleigh model is studied by Putnam [70]. A multiplicative cost estimation model in which the cost driver variables are constrained to the values (-1, 0, 1) is presented by

Walston and Felix [89]. The Doty multiplicative model constrains the cost driver variables to integers of 0 and 1 [36]. An analytical model in which the overall development effort is a function of cost driver variables is presented by Halstead [35]. Table models which relate values of cost driver variables to either a part of the overall software development effort or multipliers for effort are discussed by Aron [2]. Further discussions of available sizing and cost estimation models are available [23, 35, 59, 78, 94, 96].

Once the development size is determined the adequacy of in-house resources must be considered. Three different types of resource categories should be considered: personnel, fiscal, and equipment. The unit of measure for personnel is man-hours, for fiscal resources the unit measure is dollars, and for equipment utilization the unit measure may be memory size, speed, and other characteristics.

In the personnel resource category, three subcategories are considered: programmers, managers, and support staff. The programmers are the personnel which will be responsible for the development of the actual code required by the needs analysis statement. Three additional subcategories of programmers are

considered: experienced, intermediate, and novice. Managers may also be classified into three sub-categories: project, task, and line managers. The relationship of the managerial categories was presented in Figure 4.10. The support staff is comprised of clerks and documentation technicians necessary to maintain the paperwork associated with a development effort.

Fiscal resources are also subdivided into two categories: operational and support. The operational category is defined to contain fiscal resources that are primarily intended for actual software development, testing, and implementation. The support category identifies funds that are not directly mission related, but serve to support operational activities. Examples of this type of funding category are graphics support and telecommunications support. The fiscal resources may also be described in terms characteristic of how they will be expended. Typical fund categories include such types as operations and maintenance, procurement and research, and development test and evaluation. Each enterprise has different interpretations on how these funds may be spent, but usually these definitions are quite similar.

The equipment resource category is defined to include three different types of equipment: research

and development, operational, and support. Research and development equipment that is too costly to justify on a one-time need basis may be required to support the development effort. Specialized equipment, hardware, or other devices that would be beneficial to the overall development action are included in this category. Operational equipment includes all types of equipment listed in the software and hardware components of the software development environment (Figure 4.12 and Figure 4.13). Support equipment is not directly mission related but used to enhance the overall development action. If inadequate or no in-house equipment resources are available in one or more categories, vendor resources through leasing may be considered.

Vendor resources are listed as only people or equipment, for these are the two most salient resources that the vendor provides. The quantity of resources provided by the vendor is totally dependent upon the availability of in-house resources dedicated to a particular development project.

The decision tree of Figure 4.17 can assist in determining where various development resources may be obtained depending upon how the software project is to be completed and implemented. There are three basic steps in the vendor development support branch. If full vendor development support is required, a review of the

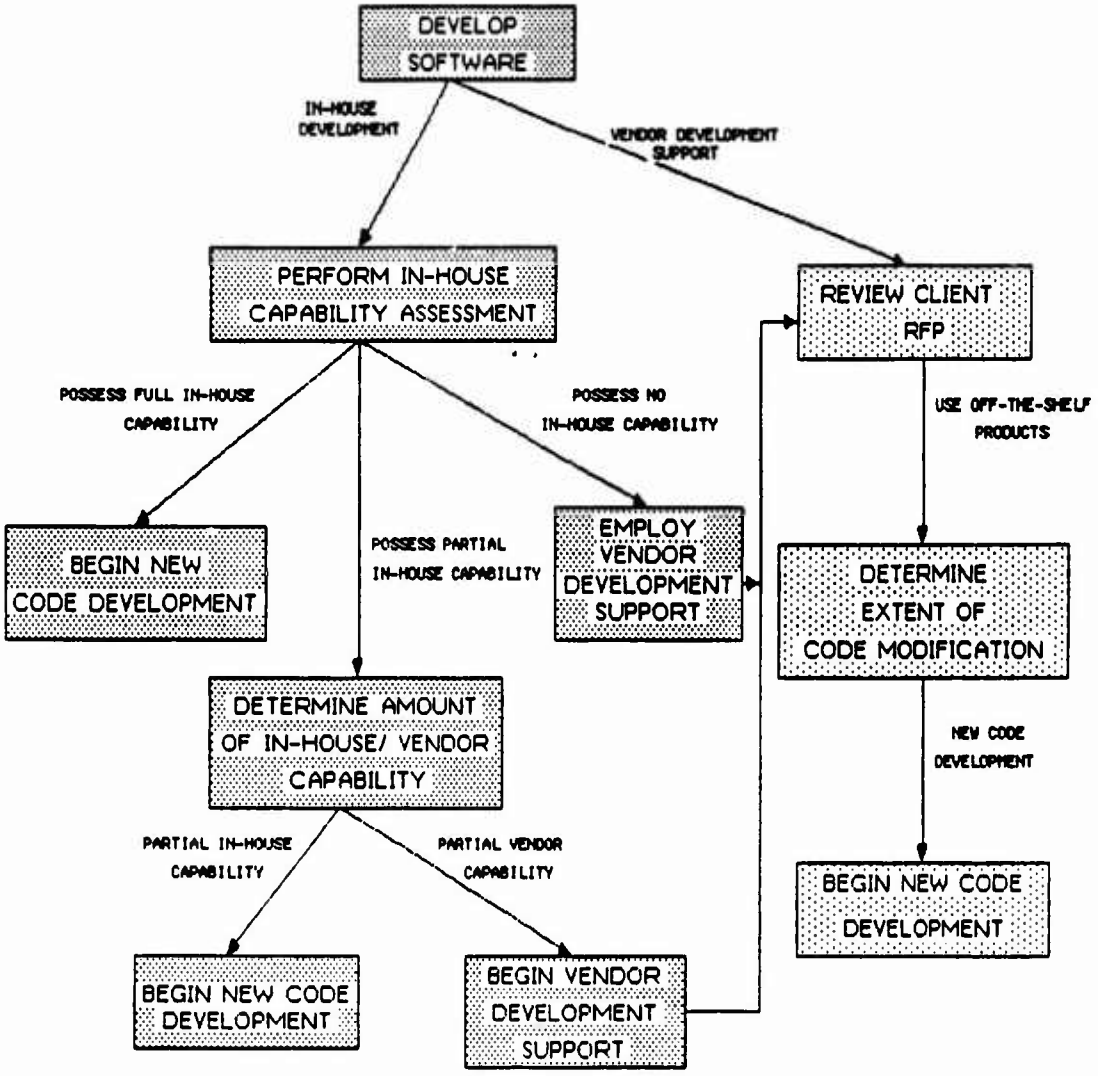


FIGURE 4.17--RESOURCE CAPABILITY DECISION TREE

request for proposal (RFP) is done to determine necessary requirements. The use of off-the-shelf products are considered in meeting software specifications. If a direct application of off-the shelf products cannot be done, the amount of modifications necessary to fulfill the specifications must be determined. If off-the-shelf and modified packages are not adequate to fulfill the specifications, then new code development is required by the vendor.

Allocation of Specific Resources

After the various job sizing methods identified in Figure 4.16 are used to determine job size, consideration is given to the adequacy of various in-house resources. For the personnel resource, lack of sufficiently qualified personnel for a major project may suggest use of external contractor support. Lack of available equipment resources may suggest that commercial leasing of a computer time sharing service may be necessary. A resource allocation matrix is developed in Chapter V which considers the various steps of software development. The matrix provides an analyst with insight on which steps require additional personnel or where large resource requirements overall are necessary. The use of the matrix will be illustrated

with the personnel resource, however, the matrix is applicable to equipment and fiscal resources also.

In-House Code Development and Integration.

After the source of development is determined (Figure 4.17), the next step is code development, testing, and integration. Figure 4.18 is the template for use with in-house developed code. The actual coding of the requirements in the specifications document is contained in the develop code block. The various interface and communications protocols are contained within the develop interfaces block. These two blocks lead to the functionality testing of the developed code.

The test plan mentioned earlier should be comprehensive and the different types of tests for the developed software identified. Since there are several types of tests commonly utilized, a brief statement about each is presented here. A more comprehensive description is available in reference [34]. The demonstration test uses controlled input to produce expected output. This test is used to show that a product has reached some level of completeness. The benchmark test certifies that the product is capable of processing input from a real user environment and producing correct results. The complete feature test

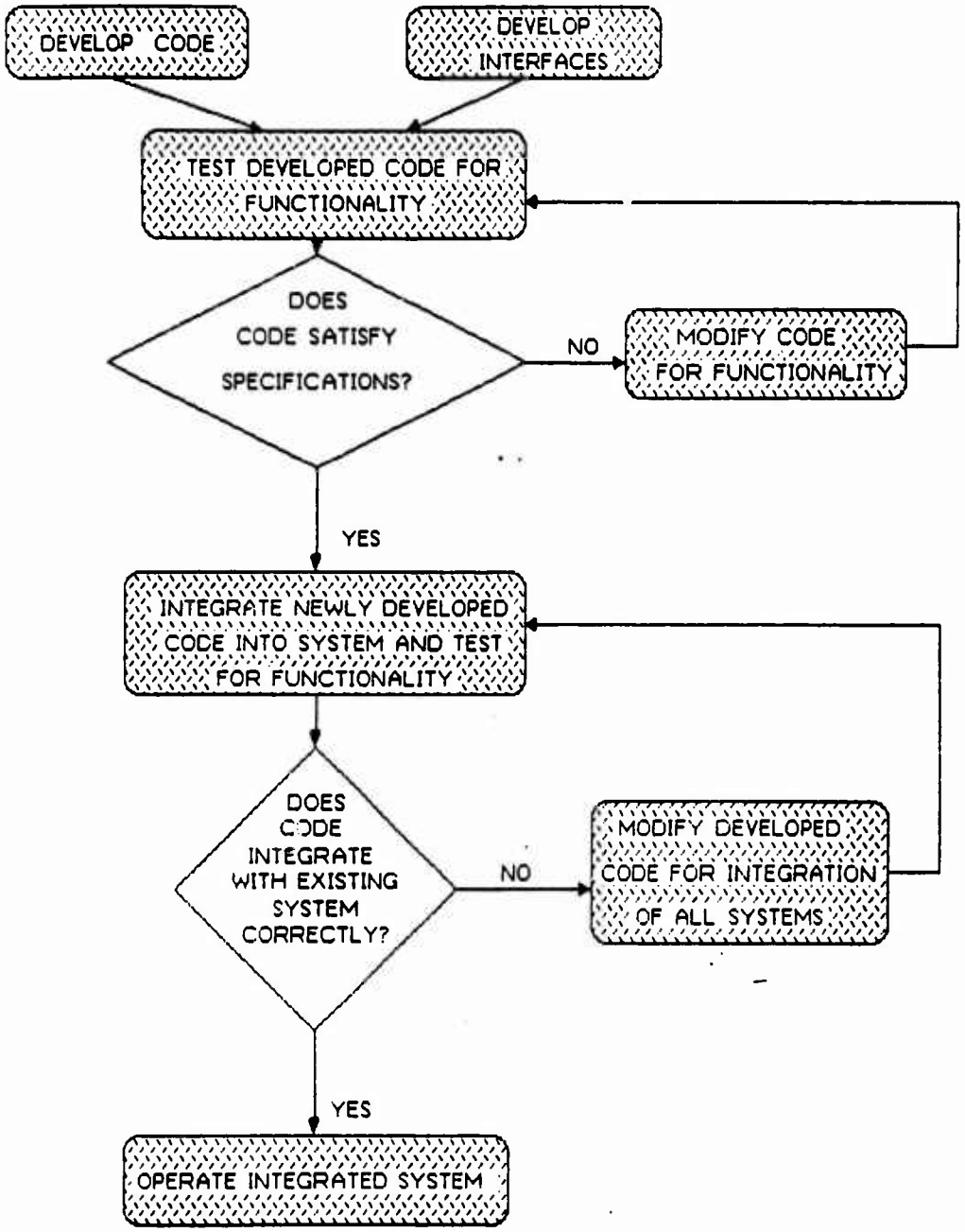


FIGURE 4.18--IMPLEMENTATION PROCEDURE FOR IN-HOUSE DEVELOPED CODE

certifies that all features specified in a product's external specifications are present and operate as described. The new feature test is used to evaluate new functions. The performance test evaluates performance characteristics such as execution speed, size, throughput, data transfer rate, compilation, assembly or generation speed, overhead, response time, and human interface. The reliability test evaluates a product under conditions that produce stress to discover its ability to operate without failure or to recover from failure such as parity errors or lost data. The stability test certifies successful integration of products into a product set or system. These are the type of tests that may be performed on the developed software before it is installed in the overall system and prior to the performance of integration tests. Testing is not totally finished until all errors are corrected. After the error correction, the newly developed code is then able to integrate with the existing system.

Vendor Development Procedure

The template in Figure 4.19 is the development procedure for vendor written code. In order to insure that the software requirements of the system

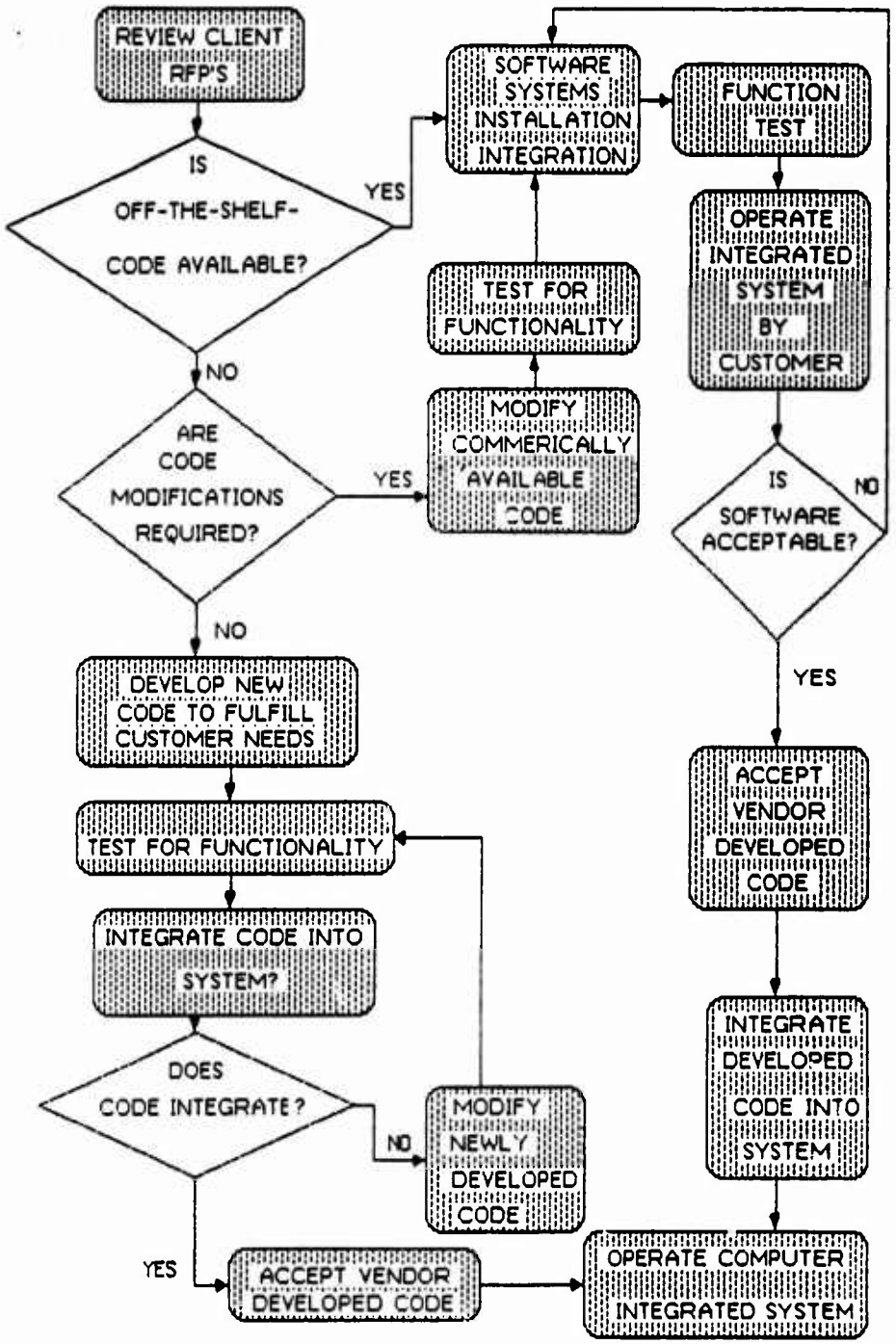


FIGURE 4.19--VENDOR DEVELOPMENT PROCEDURE

specification will be adequately fulfilled by a vendor, the use of the request for proposal (RFP) is suggested. The RFP is a contracting document that allows all vendors to provide a work submission in a format which the company can evaluate equally, insuring that a vendor does not skew their proposal for their own favor. Distinct advantages of using a RFP include [33]: detailed requirements and specifications, unbiased evaluations, consistent information, and standardized vendor proposals. The RFP should contain an objective statement, system description, proposal evaluation criteria and procedures, progress time table, user contacts, and a statement of confidentiality.

If off-the-shelf packages are available that will satisfy the systems specifications, these packages are considered first. These packages are installed and integrated into the factory environment and tested for functionality as soon as possible. The system is tested and a decision made about the acceptability of the software. If the software is not acceptable, necessary modifications are made and functionality tests again performed.

If off-the-shelf code is available and modifications are required, the modifications are made, tested, integrated, and the system operated. If new code development is required, the same tests are also

performed. The requisite protocols and interface connections must be developed independent of which method is selected.

There are many facets to the use of contract vendors for software development that are not discussed in detail here. Issues such as contract making, testing responsibility, machine usage, prototype development, patented development work, and many more must be considered and decided. After the various tests are performed and full compatibility of the code is achieved, the code is then integrated into the existing system.

Documentation and Training Requirements

A final step required before the actual developed software system can be made fully operational is the completion of documentation and training manuals for all parties who will interface with the new software (Figure 4.20). Documentation is needed not only to understand the code and how it operates, but to understand what support is required (hardware and software) so that full advantage may be taken of the various features of the newly developed product.

Documentation can be defined to be computer based and printed information that describes how the actual

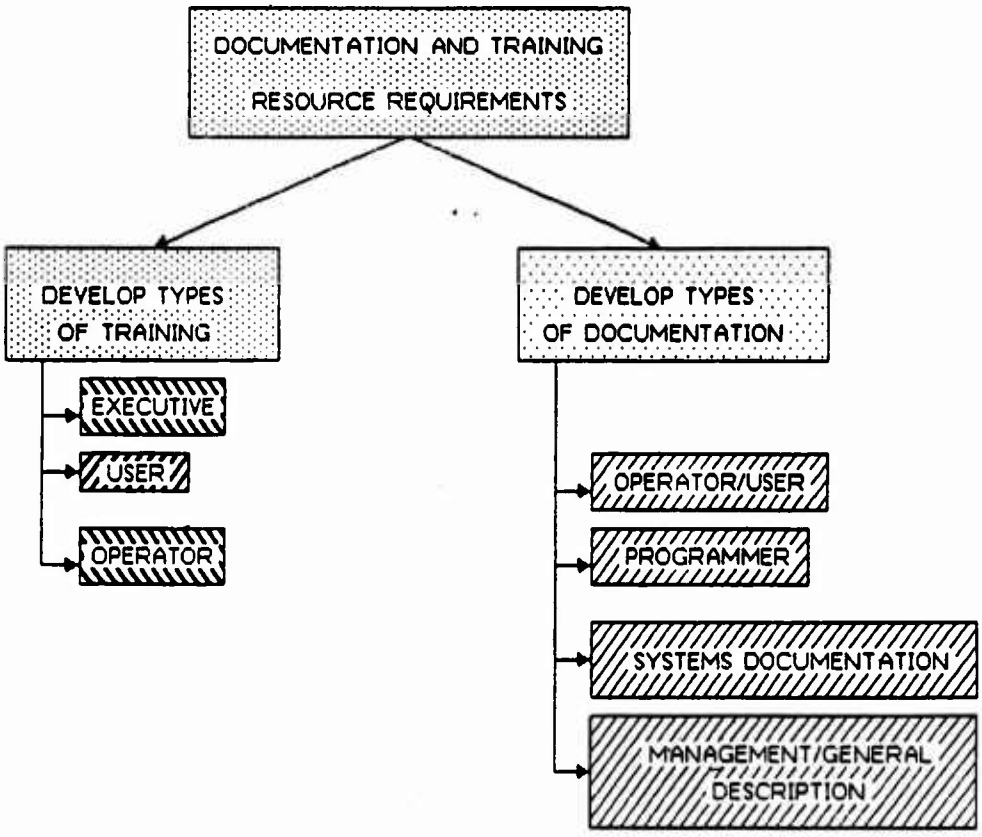


FIGURE 4.20--DOCUMENTATION AND TRAINING REQUIREMENTS

program is designed to function. This may be in a graphic, printed, or data format. Cross references should be provided in the documentation to other relevant or interfacing programs [82]. Automated documentation routines generate documentation from the source code of a program. Other automated documentation schemes provide information how data structures work, what the data is doing, and how it is used. Still other automated documentation programs provide system flow charts, system reports, and sample screens [82].

The type of documentation must be adapted to the user. Technical personnel require flow charts and system operating diagrams on the software product. Managers require overview information, while system operators and analysts need information about data types and structure, cross reference information about other (linked) programs, and the like.

Operator/user documentation can be developed using five different presentation styles [32]: prose, cookbook style, numbered instruction, playscript style, and the four-step method. The prose type of documentation tells a story to the reader on how the software performs, what inputs are required, and what actions must be taken by the reader when certain responses are necessary. The cookbook style is directive in nature: terse and succinct. This type of

documentation is good for personnel who require a regimented operational structure, but do not need to understand why each action is required. The numbered instruction method is similar to the cookbook style and provides each step broken down into numbered instructions, with sub-instructions subordinate to major instructions. The playscript style allows basically a dialogue type of presentation, with each different responsible party having his/her part identified. The four step method uses motivation, effect, general steps, and an example to illustrate the various ways in which the software product works. Further discussion on software documentation can be found in Skees [80].

Training is also required before the new system can be used to its full capacity. Different classes (Figure 4.20) can be held for executives, mid-managers, users, and system operators. The classes for executives should focus on capabilities that the system includes and how they can use executive workstations to gain greater productivity and improve managerial decision making.

Training for users of the system is different in scope and level of detail than that for the executive and focuses on the details of system features. This would include some detail and description of the various capabilities and interface data inclusion, types of calls, and procedural functions that the user requires

to effectively interface with the system in an operational environment.

Surveys which investigate the inherent knowledge of the users before the training classes commence provide for an identification of the training needs of the system users [46]. If the users understand the benefits of the new system, they more readily learn how the system will allow them to perform their job. This also allows the training staff to focus on areas with which users do not possess a familiarity.

Summary of Template Usage

This chapter has presented a discussion of the use of the various templates for planning software development activities. Different templates identify critical factors in both the technical and managerial elements of which the user of the SPD methodology must be cognizant before the overall project is initiated. A variety of templates addressing key issues of both a technical and managerial nature allow identification of the links between the currently existing software systems and those to be developed.

In manufacturing software planning and development the questions asked are the same. Many of these questions have been addressed in this chapter. However, the answers vary from one project to another.

Therefore, the frequency of use of the various templates depends upon a variety of factors. Some of the managerial templates, for example, the organizational structure, steering team or project development team, may not be used as often as templates pertaining to resource use determination. The templates used to determine whether in-house or vendor development should be pursued would probably be used most of the time that any new project is considered. The various in-house or contract implementation/integration templates would be used in each development activity. The type of the template, the size of the project, and currently used software are some of the factors which determine the use frequency.

CHAPTER V

SPD METHODOLOGY FOR PERSONNEL RESOURCE ALLOCATION MODELING

Introduction

This chapter investigates one method of allocating personnel resources to specific development steps of the software life cycle using the SPD methodology. The method investigated is applicable to personnel, equipment, and fiscal resources, however, the personnel resource has been selected for development here. The method uses the concept of resource building blocks for each step of the SPD methodology to prescribe the amount of resource required. A resource allocation matrix is combined with the respective building blocks to determine the total estimated cost of the project. Various manpower loading relationships representative of the software development industry have been selected and used to demonstrate how this approach is implemented.

Description of the Development Steps

The procedure and templates of the SPD methodology have been organized into eight progressive steps for use in the allocation of personnel and other resources. These steps are briefly defined here and the applicable

figure numbers from the SPD methodology description are identified. A more detailed explanation of the contents of the steps is presented in Appendix 1.

Step 1 is the Constraint Identification Step. This step involves the feasibility/understanding of the systems specifications of the newly proposed development project and possible technical conflicts which may inhibit successful software design. Figure 4.2 applies to this step.

Step 2 is the Generic Kernel Application Step. The generic kernels are applied to the requirements of the systems specifications to construct a functional software conceptual design. Figure 3.3 applies to this step.

Step 3 is the Software Technical Factors Step. The specific technical factors pertinent to the various generic kernels are identified. Figure 4.6 applies to this step.

Step 4 is the Kernel Application Environment Step. This step develops the functional software conceptual design into an actual application environment. Figure 3.4 applies to this step.

Step 5 is the Code Development Step. Actual software is developed from the generic kernel depiction of the application environment listed in Step 4. Some

preliminary developmental tests are conducted in this step. Figures 4.6 and 4.17 apply to this step.

Step 6 is the Test Verification/Validation/Integration Step. All of the developed modules of code are tested for operational effectiveness. Any deficiencies identified in Step 5 are corrected in this step. Figures 4.18 and 4.19 apply to this step.

Step 7 is the Systems Functionality Testing Step. This step integrates the newly developed code into the physical process which is being automated. Tests are conducted to see if the newly developed code provides the desired response identified in the requirements portion of the systems specifications document. Figures 4.18 and 4.19 apply to this step.

Step 8 is the Documentation and Training Step. This step develops the documentation and training courses needed to use the newly developed software. Figure 4.20 applies to this step.

Components of Personnel Resource Allocation Model

The personnel resource allocation procedure is composed of four components which utilize the development steps defined above.

1. Resource building blocks
2. Manpower loading relationships
3. Personnel resource allocation matrix (PRAM)
4. Cost estimation model

Resource Building Blocks

The building block approach to personnel allocation provides a structured and systematic construct which assists in the estimating, tracking, and monitoring of resource consumption and cost accumulation during each step of the SPD methodology. The resource building block approach in Figure 5.1 utilizes a three dimensional model. The estimated time to accomplish each development step is shown on the x-axis, the amount of each resource type is indicated on the y-axis, and the applicable SPD methodology template is shown on the z-axis for reference only. The measurement scale for each axis is relative to the resource type being depicted. For personnel, the number of full time equivalent (FTE) personnel, number of man-hours estimated to be expended, or other relevant measure is used for the y-axis. The height of each block is the quantity of a particular resource applied to a development step. For example, in Figure 5.1 the larger height of Block 4 signifies that more of the personnel

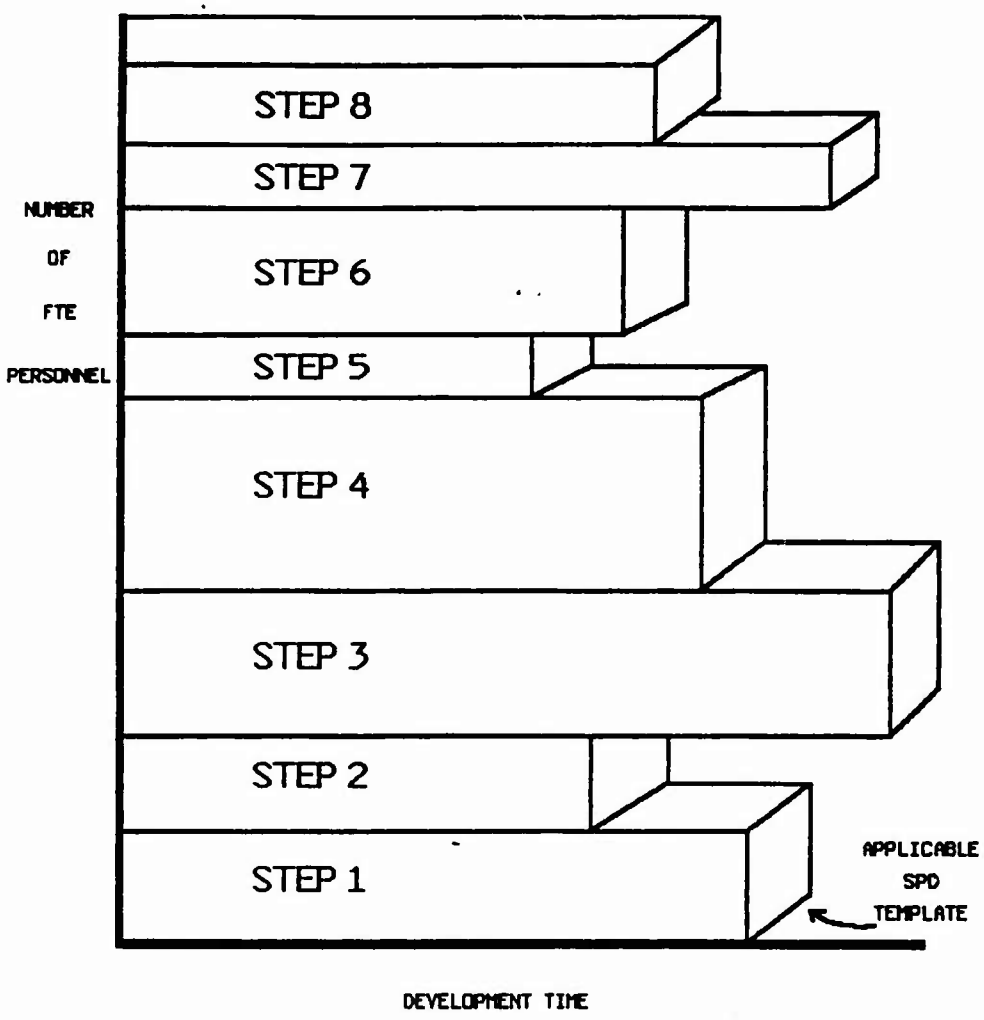


FIGURE 5.1--RESOURCE BUILDING BLOCKS

resource is needed for this step than for Step 1. The length of the blocks (x-axis) signifies the amount of time necessary to complete the development step.

For other resource types the scales could be totally different. For example, if the fiscal resource is considered, the x-axis could indicate the total amount of dollars needed in a particular development step, and the y-axis could represent the amount of funds from a specific source, such as, procurement, operations, or maintenance funds.

Manpower Loading Relationships

For the manpower loading requirements, three types of personnel resources are normally used in software development: programmers, managers, and support staff. There are definable subcategories of each type. Each personnel type may be generally allocated via a model for the individual steps of the development process. Thus, in one step the support staff may be the prime necessary resource with several other resources assigned in some proportion to the time allocation of the staff. In another step, the programmer may be the prime resource needed. The following resource types are used in this presentation:

X_1 = Novice Programmer

X_2 = Intermediate Programmer

X_3 = Experienced Programmer

X_4 = Maintenance Programmer

X_5 = Project Manager

X_6 = Line Manager

X_7 = Task Manager

X_8 = General Manager

X_9 = Documentation Librarian

X_{10} = Clerical Personnel

X_{11} = Engineering Staff

The three categories of personnel resources can be identified in these definitions: programmers (X_1 through X_4; managers (X_5 through X_8); and support staff (X_9 through X_{11}). Depending upon which development step is considered, different personnel types are utilized, that is, not every personnel type is used in each software development step.

The manpower loading relationships used in this work are all linear. Although the proportions of the various resource types may vary throughout the development steps, each varies in the same proportion as its relationship to the prime personnel resource.

The manpower loading relationships used in this research were obtained from a commercial industry source involved in software engineering [22]. These

relationships are heuristically based, but provide a good reference point of how various personnel rsources are allocated to software development projects. These relations are applicable for moderate to large development efforts.

The first development step, Constraint Identification, has a loading relationship identified as:

$$2X_5 + 1X_8 + 1X_{10} + 30X_{11} \quad (5.1)$$

This relationship states that for every thirty hours of personnel type X_{11} (engineering staff) there should be one hour each of types X_{10} and X_8 and two hours of type X_5 assigned to accomplish the step. Another interpretation is that type X_{11} is applied at a rate thirty times greater than X_8 and X_{10} and fifteen times greater than X_5 in Step 1. Each of the eight development step loading relationships are further discussed in Appendix 1.

Personnel Resource Allocation Matrix (PRAM)

The PRAM is used to assign personnel resources to the development steps. The matrix format (Figure 5.2) lists the development steps S_j across the columns (j = 1, 2, ..., 8) and the types of personnel resources X_i down the rows (i = 1, 2, ..., 11). The entries in the

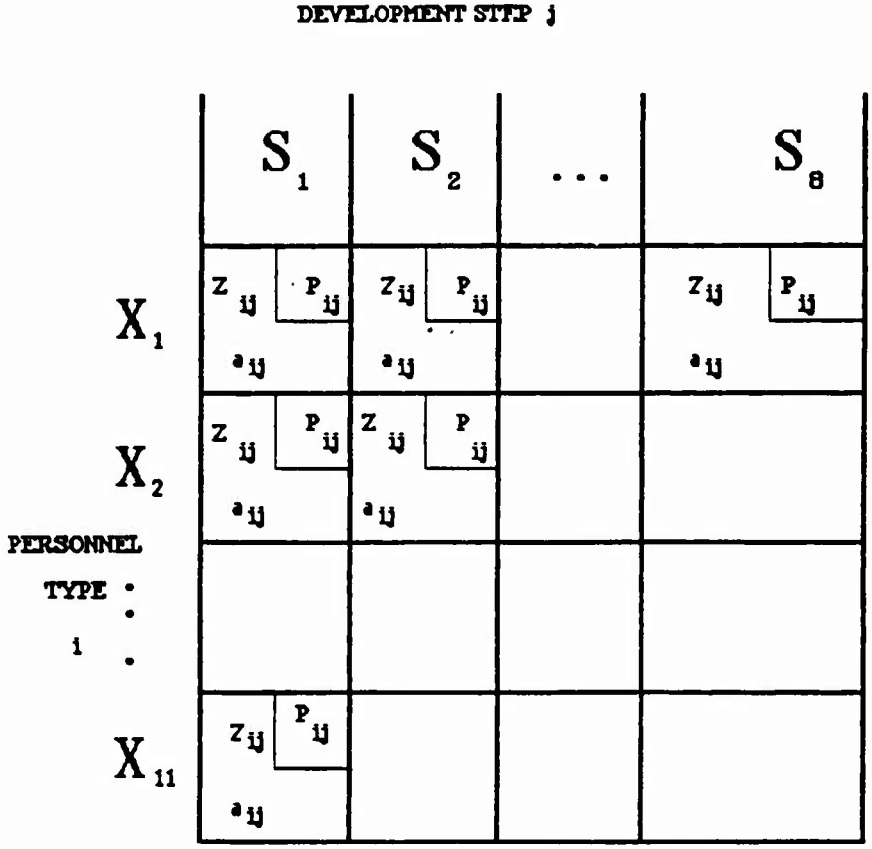


FIGURE 5.2--PERSONNEL RESOURCE ALLOCATION MATRIX (PRAM)

PRAM cells include the following:

Z_{ij} = number of full time equivalent (FTE) personnel of type i assigned to step j.

P_{ij} = manpower loading relation coefficient applicable to personnel type i for step j, in hours.

a_{ij} = maximum FTE personnel of type i available for allocation to step j.

($Z_{ij} >= 0$, $P_{ij} >= 0$, $a_{ij} >= 0$)

If no entries are present in a cell, the step does not normally need this personnel type for its completion.

The values of a_{ij} and Z_{ij} entered into the PRAM are dependent upon the size of the P_{ij} for the cell. If there are available the number of hours indicated by the P_{ij} coefficient, the value $a_{ij} = 1.0$ is present in the cell, whereas twice the number of hours of the coefficient means that $a_{ij} = 2.0$ is correct.

The P_{ij} values used in this research are generally applicable for the personnel resource to a variety of software development projects and do not need to be re-defined for each type of project. If the size of the software project is judged to require about the number of man-hours equal to the P_{ij} in the loading relation, $Z_{ij} = 1$ is entered for all cells where there is a positive P_{ij} present. If, however, the software tasks for the specific development step j are expected

to require, for example, two times the indicated P_{ij}, the entry is $Z_{ij} = 2$. The other Z_{ij} values are entered accordingly using the relative weightings of the P_{ij} values. As an illustration, if a project planner is utilizing relation (5.1) for personnel planning in Step 1, and believes the effort will require 120 hours of engineering staff time (X_{11}), the entry will be $Z_{11,\ 1} = 4.0$. To correctly use the guideline of relation (5.1), the entries for all Z_{i1} will be 4.0. The planner may wish to deviate from the guidelines, or include the use of personnel types not included in the loading relation. This departure from the guideline relation is accounted for in the personnel cost estimation model presented below.

If in a particular application of the PRAM, $(Z_{ij} - a_{ij}) > 0$, there is an indication that the in-house personnel level is insufficient to support the requirements of the development step. The use of external contract personnel is necessary if the Z_{ij} is a realistic estimate of the personnel needs.

If, on the other hand, $(a_{ij} - Z_{ij}) > 0$, the manager has excess personnel resources for step j and has the option of redistribution of personnel to other cells. The reallocation would normally be done within the same category, that is, programmers, managers, or support staff. In this manner, deficiencies within a

particular resource category can be eliminated by similar personnel types.

The value of Z_{ij} assigned to a cell of the PRAM is dependent upon the judgemental experience of the manager and the size of the loading coefficient P_{ij}. A large value of P_{ij} indicates a large need of that particular type of resource. Additionally, other resource types should be allocated in a reasonable proportion according to the P_{ij} values in the loading relation for the step. Completion of each step of the PRAM is independent. However, interchangeability of personnel types between steps provides flexibility in the matching of the resource availability with that estimated to be required by the software development manager.

A PRAM with the P_{ij} entries from the manpower loading relationships of the eight development steps is presented in Figure 5.3.

Cost Estimation Model

The estimated total personnel cost TC of the software development project may be written as

$$TC = \sum_{i=1}^{11} \sum_{j=1}^{8} IC_i Z_{ij} P_{ij} + \sum_{i=1}^{11} \sum_{j=1}^{8} OC_i (Z_{ij} - a_{ij}) P_{ij} \quad (5.2)$$

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
X_1					100	100	10	100
X_2					100	100	10	100
X_3					100	100	10	100
X_4								
X_5	2	5	2	1		9	6	3
X_6		10	4	2	6	12	12	6
X_7		40	20	20	5	45	30	45
X_8	1	1				6		
X_9		20					180	180
X_{10}	1	100	20		120			
X_{11}	30		50	100	60	60	90	90

FIGURE 5.3--PERSONNEL RESOURCE ALLOCATION MATRIX WITH P_{ij} COEFFICIENTS

where IC_i = cost per man-hour of in-house personnel resource i

OC_i = cost per man-hour of external contract personnel of type i

The values for IC and OC may be obtained from company sources, national surveys, and quotations from external vendors. The values should include base salary plus benefits and normally associated overhead costs for the type of resource.

If no in-house personnel are available for the project, the first double summation term of equation (5.2) is zero. Hence the complete project would require contract development. As mentioned in Chapter III, being able to carefully identify how the project workload is apportioned assists the in-house contract monitor in determining the total cost for contracted tasks.

The resource allocation matrix cost estimation capability could be easily adapted to a spread sheet format for more rapid data manipulation. Some layering of current spread sheets would provide enhanced identification of resource short falls and also allows a greater sensitivity analysis to be performed on the various cost estimates. This capability allows in-house personnel to evaluate various cost estimates before determining the optimum strategy.

Other Resource Allocation Methods Considered

Several significant operations research algorithms to include simulation, dynamic, and integer programming [67] were investigated for possible applicability to this research. In all cases, however, the lack of the necessary data or implicit assumptions of the various models prohibited adaptation to the research effort.

A linear programming resource allocation problem formulation was explored in detail. The objective function was stated to be a minimization of the personnel resource costs, with the constraint equations to be various manpower loading relationships coupled with an appropriate man-hour limit for each relationship. Dimensionability of variables became a significant problem that was difficult to resolve and continue to maintain implicit model assumptions. Because the objective function was a minimization, all constraint equations had inequalities of a less than or equal to nature. The obvious solution was zero resources applied. A dual problem formulation was attempted, trying to somehow quantify human productivity. No robust manner was found in which to logically measure and equate human productivity for resource allocation. Since no convex hull could be developed that had practical measurable parameters, a

linear programming formulation was determined to be infeasible.

Another problem identified was that the linear programming model did not address the possibility of external resources being required if insufficient in-house resources were available. For these reasons, the resource allocation matrix was developed and used.

CHAPTER VI

CONCLUSIONS

This research has developed a planning and development methodology for the design, implementation and cost estimation of large software projects. The overall approach of systems engineering is applied to a problem which has proven to be unwieldy, not only from a managerial and technical perspective, but from a resource estimation and allocation viewpoint.

Results of the Research

There are several major results of the research included in the SPD methodology. A summary of the more important ones is given here:

1. A high level, strategic planning methodology for software is developed which integrates technical and managerial elements with an approach to resource use determination. The methodology, developed as a parallel to the classic software life cycle provides identification and coordination of the essential facets identified with software engineering.

2. The concept of the kernel construct is presented and demonstrated as an approach to

functionally planning a large software engineering/ development effort. The kernel construct is shown in the generic form and in a form applicable to the software design of a computer integrated manufacturing environment. Each software project is comprised of different types of kernels with each kernel related to one type of function.

3. Generic kernels are tailored to specific improvements through the use of templates, which address the significant technical, managerial, and resource determination factors pertinent to software system planning. Each template includes a flowchart, decision chart, or structural diagram useful in the information collection and planning stages of a development project. Each template is linked to specific blocks in the SPD methodology.

4. The resource allocation function includes the coordinated use of templates and a building block approach to software systems planning. A part of this approach is the use of a personnel resource allocation matrix (PRAM). The PRAM makes use of linear manpower loading relationships which characterize different types of manpower necessary for software development. These relations have been determined and applied heuristically and provide an idea of how manpower could be allocated to the various steps of software development.

5. An ability to provide personnel resource cost estimates for software development is possible by utilizing the PRAM and costs for organization organic and external contract personnel. Although only the uses of personnel resources is detailed, the development of the approach using the methodology for fiscal and equipment resources may be possible using a similar method.

Recommendations for Further Work

This work has developed the kernel construct for the strategic planning level. Additional work can be initiated on developing kernels that are at the tactical and operational planning levels. These lower level kernels would focus on less breadth and more depth for technical and managerial planning levels. The overall result could be a complete kernel system that would be used to assist in developing and implementing operational software.

Investigation of the different types of fiscal resources necessary to fund the development of software may be undertaken. This investigation would address the breakout of funds by proportion from procurement/ acquisition, research/development, and operations/ maintenance sources. The concept of a resource

allocation matrix could be adapted to provide a graphic pictorial of the amounts of money necessary to accomplish the different facets of the development work. Specific fiscal resource allocation equations similar to the personnel loading relationships may be developed for fiscal categories.

Additional research is warranted in using the SPD methodology and resource allocation process for software development for areas other than the manufacturing environment. Software development for the financial services and construction sectors, for example are areas where the SPD methodology could have substantial applicability.

Investigation as to whether the SPD methodology can be adapted to environments other than the software design is warranted. An environment in which the managerial and relevant technical elements are best understood and coordinated through an organized resource planning and allocation procedure may benefit from this approach.

REFERENCES

1. Ahituv, Niv, I. Borovits and Z. Pomeranz, "Managing Large Sub Contracted Projects," Journal of Information Systems Management, Vol. 2, No. 3, Summer 1985, pp. 27-35.

2. Aron, J. D., Estimating Resources for Large Programming Systems, NATO Science Committee, Rome, Italy, October 1969.

3. Baker, F. T., "Chief Programmer Team Management of Production Programming," IBM Systems Journal, Vol. 11, No. 2, 1972, pp. 56-73.

4. Baker, F. T., "Structured Programming in a Production Programming Environment," Proceedings, International Conference on Reliable Software, IEEE Computer Press, Los Angeles, California, 1975, pp. 172-183.

5. Basili, Victor R., Models and Metrics for Software Management and Engineering, IEEE Computer Press, Los Alamitos, California, 1980.

6. Bell, T. E., et al., "An Extendable Approach to Computer-Aided Software Requirements Engineering," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, January 1977, pp. 49-60.

7. Bender, P. S. et al., "Practical Modelling for Resource Management," Harvard Business Review, Vol. 59, No. 2, March-April 1981, pp. 163-173.

8. Blanchard, Benjamin S. and W. J. Fabrycky, Systems Engineering and Analysis, Prentice Hall, Englewood Cliffs, New Jersey, 1981.

9. Blank, Leland T. and H. Carrasco, "System Development Methodologies," Report for ICAM Technology Transfer, United States Air Force, Department of Industrial Engineering, Texas Engineering Experiment Station, College Station, Texas, 1984.

10. Boehm, Barry W., Software Engineering Economics, Prentice Hall, Englewood Cliffs, New Jersey, 1981.

11. Boehm, Barry W., "Software Engineering," IEEE Transactions on Computers, Vol. C-25, No. 12, December 1976, pp. 1226-1241.

12. Brooks, F. P., Jr., The Mythical Man Month, Addison Wesley, Reading, Mass., 1975.

13. Buckle, J. K., Managing Software Projects, American Elsevier, New York, New York, 1977.

14. Caine, S. H. and E. K. Gordon, "PDL-A Tool for Software Design," AFIPS Conference Proceedings National Computer Conference, Vol. 44, No. 1, 1975, pp. 223-234.

15. Carlin, J. W., "A Steering Committee for Equal Representation," Management World, Vol. 7, No. 4, April 1978, pp. 32-33.

16. Chorafas, Dimitris N., The Software Handbook, Petrocelli Books, Princeton, New Jersey, 1984.

17. Cooper, Jack, "Software Development Management Planning," IEEE Transactions on Software Engineering, Vol. SE-10, No. 1, January 1984, pp. 22-26.

18. DeMaagd, Gerald R., "Matrix Management," Datamation, Vol. 14, No. 10, October 1970, pp. 100-103.

19. Devenney, T. J., An Exploratory Study of Software Cost Estimating at ESD, GSM/SM/76S-4: Thesis, Air Force Institute of Technology, Wright Patterson AFB, Ohio, July 1976.

20. Devert, Ernst, "The Project Library--A Tool for Software Development," Proceedings, 4th International Conference on Software Engineering, Munich, Germany, September 17-19, 1979, pp. 153-163.

21. Drury, D. H., "A Survey of DP Steering Committees," Information and Management, Vol. 9, No. 1, August 1985, pp. 1-7.

22. Dudic, Mary, personal letter, April 15, 1986, Honeywell Sea Systems, Seattle, Washington.

23. Elshoff, J. L., "An Investigation into the Effects of the Counting Method Used on Software Science Methods," ACM SIGPLAN NOTICES, Vol. 13, No. 2, February 1978, pp. 29-46.

24. Elshoff, J. L., "An Analysis of Some Commercial PL-1 Programs," IEEE Transactions on Software Engineering, Vol. C-25, No. 6, June 1976, pp. 113-120.

25. Ford, F. Nelson, et al., "The Evolving Factory of the Future: Integrating Manufacturing and Information Systems," Information and Management, Vol. 8, No. 8, 1985, pp. 76-102.

26. Freiman, F. R. and R. D. Park, "Price Software Model Version 3--An Overview," Proceedings IEEE PINY Workshop on Quantitative Software Models, IEEE Catalog TH-0067-9, October 1979, pp. 32-41.

27. Gaffney, J. E. Jr., "The Impact of Software Development Costs Using HOL's," IEEE Transactions on Software Engineering, Vol. SE-12, No. 3, March 1986, pp. 496-499.

28. Gane, C. and T. Sarson, Structured Systems Analysis, Prentice Hall, Englewood Cliffs, New Jersey, 1979.

29. Gilbert, Philip, Software Design and Development Science Research Associates, Chicago, Illinois, 1983.

30. Gilhooley, Ian A., "A Methodology for Productive Software Development," Journal of Information Systems Management, Vol. 3, No. 1, winter 1986, pp. 36-41.

31. Griffin, Ricky, Management, Haughton-Mifflin, Boston, Massachusetts., 1984.

32. Grimm, Susan J., How to Write Computer Manuals for Users, Lifetime Learning Publications, Belmont, California. 1982, pp. 50-55.

33. Guerrieri, John A., Jr. "How to Develop Effective Request for Proposals," Journal of Information Systems Management, Vol. 1, No. 4, Fall 1984, pp. 40-47.

34. Gunther, Richard C., Management Methodology for Software Product Engineering, John Wiley and Sons, New York, New York, 1978, pp. 133-137.

35. Halstead, M. H., Elements of Software Science, Elsevier-North Holland, New York, New York, 1977.

36. Herd, J. R. et al., Software Cost Estimation Study--Study Results, Final Technical Report, RADC-TR-77-220, Vol. I, Doty Associates, Inc., Rockville, Maryland, June 1977.

37. Huggins, Lawrence P., "Robotics The Coming Challenge for MIS Managers," Journal of Information System Management, Vol. 2, No. 4, Fall 1984, pp. 3-7.

38. Huse, Edgar, T., Management, West Publishing Company, Saint Paul, Minnesota, 1982, pp. 262-266.

39. Hymowitz, Carol, "Manufacturing Change: A Special Report on Technology in the Workplace," Wall Street Journal, Section 3, September 16, 1985, pp. 10C-12C.

40. Jackson, M. A., Principles of Program Design, Academic Press, New York, New York, 1975.

41. Janossy, James G., Commercial Software Engineering, John Wiley and Sons, New York, New York, 1985, pp. 9-10.

42. Kircher, Paul and R. O. Mason, Introduction to Management: A Systems Approach, Melville Publishing Company, Los Angelos, California, 1975, p. 94.

43. Koontz, Harold and C. O'Donnel, Management, McGraw-Hill Book Co., New York, New York, 1976, p. 309.

44. Kriendler, Jerry, "Contracting for the Development of Customized Software," The SCOTT Report, Vol. 4, No. 9, September 1985, pp. 1-12.

45. Lardner, James, Unpublished ICAM Industry Days address, New Orleans, Louisiana, May 1982.

46. Ledbetter, William N., et al., "Education and Training Needs Must be Assessed Before Systems Implementation," Information Management, Vol. 24, No. 5, May 1986, pp. 16-19.

47. Levulis, R. J., "CIM--A Perspective," Current Awareness Bulletin-- Manufacturing Technology Information Analysis (MTIAC), Vol. 1, No. 1, Spring 1985, p. 3.

48. Lewis, T. G., Software Engineering--Analysis and Verification, Reston Publishing Co., Reston, Virginia, 1982.

49. Lundeberg, M., et al., Information Systems Development--A Systematic Approach, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

50. Lundeberg, M., et al., "A Systematic Approach to Information Systems Development," Information Systems, Vol. 4, No. 1, 1979. pp. 1-12

51. Lundeberg, M., et al., "A Systematic Approach to Information Systems Development," Information Systems, Vol. 4, No. 2, 1979, pp. 93-118

52. Lundeberg, M., "An Approach for Involving the Users in the Specification of Information Systems," Formal Models and Practical Tools for Information Systems Design, Schneider, Amsterdam, North Holland, 1979, pp. 195-217.

53. McKeen, J. D., "Activity Analysis: An Approach to Understanding the System Development Process," Proceedings ASAC 1982 Conference, University of Ottawa, Ottawa, Canada, 1982, pp. 41-50.

54. Metzger, P. J., Managing a Programming Project, Prentice Hall, Englewood Cliffs, New Jersey, 1973.

55. Metzner J. R., "A Graded Bibliography on Macro Systems and Extensible Languages," SIGPLAN NOTICES, February 1979.

56. Meyers, G. J., Composite/Structure Design, Van Nostrand Reinhold, New York, New York, 1978.

57. Meyers, G. J., Software Reliability, John Wiley and Sons, New York, New York, 1976, pp. 220-246.

58. Mize, Joe H., et al., "Strategic Planning for Factory Modernization: A Case Study," National Productivity Review, Winter 1984-85, pp. 33-44

59. Mohanty, Siba N., "Software Cost Estimation: Present and Future," Software Practice and Experience, Vol. 11, No. 2, February 1981, pp. 103-121.

60. Moranda, P. B., "Software Quality Technology: Status Of, Limits To, Alternative To," IEEE Computer Society, Computer Magazine, Vol. 13, No. 11, November 1978, pp. 72-78.

61. Nassi, I. and B. Shneiderman, "Flowchart Techniques for Structured Programming," ACM SIGPLAN NOTICES, Vol. 8, No. 8, August 1973, pp. 12-26.

62. Nelson, E. A., Management Handbook for Estimating Computer Programming Costs, System Development Corporation, Boston, MA., October 31, 1966.

63. Parikh, Girish, "How to Pick a Winner, A Look at Software Methodologies," ACM SIGSOFT, Software Engineering Notes, Vol. 8, No. 2, April 1983, pp. 33-39.

64. Parker, M. M., "Enterprise Information Analysis: Cost Benefit Analysis and the Data Managed System," IBM Systems Journal, Vol. 21, No. 1, 1982, pp. 108-121.

65. Parnas, D. L., "Designing Software for Ease of Extension and Contraction," IEEE Transactions on Software Engineering, Vol. SE-5, No. 3, March 1979, pp. 128-138.

66. Peters, L. J., Software Design: Methods and Techniques, Yourdon Press, New York, New York, 1981.

67. Phillips, Don T., et al., Operations Research, Principles and Practice, John Wiley and Sons, New York, New York, 1976.

68. Pressman, Roger S., Software Engineering, McGraw Hill, New York, New York, 1982.

69. Prince, T. R., Information Systems for Management Planning and Control, R. D. Irwin, Homewood, Illinois, 1975.

70. Putnam, L. H., "A General Estimating Solution to the Macro Software Sizing and Estimating Problem," IEEE Transactions on Software Engineering, Vol. SE-4, No. 1, July 1978, pp. 345-361.

71. Raunch-Hindin, Wendy, "Flexible Automation," Systems and Software, Vol. 4, No. 12, December 1985, pp. 28-37.

72. Reynolds, William H., "The Executive Synecdoche," MSU Business Topics, Vol. 17, No. 4, Autumn 1969, p. 26.

73. Robey, D. and D. Farrow, "User Involvement in Information Systems Development: A Conflict Model and Empirical Test," Management Science, Vol. 28, No. 1, 1982, pp. 73-85.

74. Ross, D. T., "Reflections on Requirements," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, January 1977, pp. 110-115.

75. Ross, D. T., and K. E. Schoman, Jr., "Structured Analysis for Requirements Definition," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, January 1977, pp. 69-84.

76. Ross, D. T., "Structured Analysis (SA): A Language for Communicating Ideas," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, January 1977, pp. 16-34.

77. Sammett, J., "Roster of Programming Languages for 1976-1977," SIGPLAN NOTICES, November 1978.

78. Schneider, V., "Prediction of Software Effort and Project Duration--Four New Formulas," SIGPLAN NOTICES, Vol. 13, No. 6, June 1978.

79. Shooman, M. L., Software Engineering: Design, Reliability and Management, McGraw Hill, New York, New York, 1983, p. 122.

80. Skees, William D., Writing Handbook for Computer Professionals, Lifetime Learning Publications, Belmont, California, 1982.

81. Snyder, Charles A. and J. F. Cox, "A Dynamic Systems Development Life Cycle Approach: A Project Management Information System," Journal of Management Information Systems, Vol. II, No. 1, Summer 1985, pp. 61-76.

82. Snyders, Jan, "Blueprinting Systems for Better Productivity," Infosystems, Vol. 32, No. 12, December 1985, pp. 40-42.

83. Stay, J. F., "HIPO and Integrated Program Design," IBM Systems Journal, Vol. 15, No. 2, 1976, pp. 143-154.

84. Stuart, Walter J., "An Experiment in DP Management--Revisited," Datamation, Vol. 13, No. 11, November 1969, pp. 149-157.

85. Terchroew, D. and E. A. Hershey, "Structured Documentation and Analysis," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, January 1977, pp. 23-30.

86. Thibodeau, R., "An Evaluation of Software Cost Estimation Models," General Research Corporation, Report T10-2670, Boston, MA., April 1981.

87. Trainor, W. L., "Software: From SATAN to SAVIOR," Proceedings, NAECON, Los Angeles, CA., May 1973.

88. Umbaugh, Robert E., "How to Make the Most of an MIS Steering Committee," Journal of Information Systems Management, Vol. 1, No. 3, Summer 1984, pp. 13-19.

89. Walston, C. E. and C. P. Felix, "A Method of Programming Measurement and Estimation," IBM Systems Journal, Vol. 16, No. 1, January 1977, pp. 54-73.

90. Walston, C. E. and C. P. Felix, "A Method of Programming Measurement and Estimation," IBM Systems Journal, Vol. 16, No. 1, January 1977, pp. 64-65.

91. Wasserman, Anthony I., Software Development Environments, IEEE Computer Society Press, Los Almitos, California, 1981, pp. 15-35.

92. Wasserman, A. I. and S. K. Stinson, "A Specification Method for Interactive Information Systems," Proceedings: Specifications of Reliable Software, IEEE Computer Society, 1979, pp. 68-79.

93. Wirth, N., "Program Development by Stepwise Refinement," Communications of the Association for Computing Machinery, Vol. 14, No. 4, April 1971, pp. 221-227.

94. Wolverton, Ray W., "The Cost of Developing Large Scale Software," IEEE Transactions on Computers, Vol. C-23, No. 6, June 1974, pp. 615-636.

95. Yau, Stephen S., and J. J. P. Tsai, "A Survey of Software Design Techniques," IEEE Transactions on Software Engineering, Vol. SE-12, No. 6, June 1986, pp. 24-44.

96. Yourdon, Edward Nash, Classics in Software Engineering, Yourdon Press, New York, New York, 1979.

97. Yourdon, E. and L. L. Constantine, Structured Design, Van Nostrand Reinhold, New York, New York, 1979.

98. Zelkowitz, Marvin V., et al., Principles of Software Engineering and Design, Prentice Hall, Englewood Cliffs, New Jersey, 1979.

APPENDIX 1

Manpower Loading Relationships

The following manpower loading relationships are taken from an industrial source [22]. Each organization or company may have its own set of relationships, but those presented here may typically represent how manpower is applied to software development projects.

There are several types of personnel categories utilized in software development projects. For the purposes of resource allocation, the applicable types are defined here and related through the manpower loading relationships. There are eleven different types of personnel resources:

X_1 = Novice Programmer

X_2 = Intermediate Programmer

X_3 = Experienced Programmer

X_4 = Maintenance Programmer

X_5 = Project Manager

X_6 = Line Manager

X_7 = Task Manager

X_8 = General Manager

X_9 = Documentation Librarian

X_{10} = Clerical Personnel

X_{11} = Engineering Staff

Three categorizations of personnel resources can be identified. The programmers, (X_1 through X_4) are the first category, the management staff (X_5 through X_8) are the second category and the support staff (X_9 through X_{11}) are the third category. Depending upon what step of the methodology is being considered determines which personnel types are employed. It is not necessary to use each personnel type in each software development step. . .

In the development of software, there are several sequential steps that must be executed before operational code is realized. Although different code development processes, methodologies, and procedures use different names for the various steps, the function of each step is similar. For the purpose of this research each of the development steps will be defined. There is one key personnel resource that is considered to be the "prime" resource for the development step. The allocation of respective manpower for each step will focus on the proportionality of the key resource to all other resource types in that step.

Step 1 is the Constraint Identification Step. This step involves the feasibility/understanding of tne systems specifications of the newly proposed development project. The objective of this step is for the analyst to possess a clear understanding of the problem to be

solved and how the developed software will perform that task. Figure 4.2 applies to this step.

Step 2 is the Generic Kernel Application Step. This step defines the accepted baseline definition of the user/mission requirements or system specifications. This understanding of the overall system requirements is translated into a form compatible with the specified design processes and the requisite hardware through use of the various generic kernels. The identification of interface protocols and system performance testing requirements are identified during this step. Figure 3.3 applies to this step.

Step 3 is the Software Technical Factors Step. This step includes detailed analysis and concept designs through use of the generic kernels to determine the operational/performance requirements for the hardware and software, trade offs between hardware and software capabilities and economic benefit, system loading and timing, and interface protocol requirements between hardware and software. Figure 4.6 applies to this step.

Step 4 is the Kernel Application Environment Step. This step develops the software design to the point where all of the kernel interfaces are designated. Orders of hierarchy for the various kernels, code modules within each kernel, data base design, inputs to and outputs from each module and kernel along with

understanding of protocols would be addressed in this step. Figure 3.4 applies to this step.

Step 5 is the Code Development Step. This step converts the generic kernel depiction of the application environment to actual software. Depending upon decisions made in the software technical factors template, (Figure 4.6) determines the type of language the code is written in. Some preliminary developmental test are also conducted in this step. Figures 4.6 and 4.17 apply to this step.

Step 6 is the Test Verification/Validation/Integration Step. All of the various modules of code developed in Step 5 are tested for operational effectiveness. Deficiencies identified during the preliminary developmental tests are corrected in this step. The initial loading of the developed software to operate and control physical devices is done in Step 6. Figures 4.18 and 4.19 apply to this step.

Step 7 is System Functionality Testing. This step integrates the newly developed software into the physical process which is being automated. Testing is conducted to insure that the newly developed software provides the desired response that was stated in the systems specifications document. Design errors identified in this step are resolved in Step 6 and

testing re-accomplished. Figures 4.18 and 4.19 apply to this step.

Step 8 constructs the documentation and training of the newly developed code. Training is conducted at all levels which will have any interface with the new code. Documentation is usually prepared as the code is being developed. New self documenting software development languages alleviate a great burden of the documentation step. Figure 4.20 applies to this step.

The various manpower loading relationships for each of the steps are given below.

In Step 1, the key resource is X_{11}, the engineering staff. The specific relationship is as follows:

$$2X_5 + 1X_8 + 1X_{10} + 30X_{11}$$

The interpretation of the relationship states that for every thirty hours of resource type X_{11} that is applied to a software development project, one hour of resource type X_{10}, one hour of resource type X_8, and two hours of resource type X_5 should be applied to the project. Another way to look at this relationship is the following: For this particular step, resource type X is applied at a rate thirty times greater than either resource types X_8 or X_{10} and fifteen times greater than resource type X_5. The interpretation of the remaining manpower loading relationships is just the

same as that given above. The only difference is the number of resource types that are contained within the respective manpower loading relationship.

Although only manpower loading relationships for personnel are discussed in this research, the concept remains valid for other types of resources. The proportionality of resource types does not clearly have to be linear. Square root, cube root, and exponential coefficient relationships can also be considered valid coefficient types depending upon the particular resource category.

The particular manpower loading relationship for Step 2 is:

$$5X_5+10X_6+40X_7+1X_8+20X_{10}+100X_{11}$$

The prime resource in this step is resource type X_{11}, the engineering staff. The composition of the engineering staff includes software engineers, electrical and mechanical engineers, along with software system analysts.

The particular manpower loading relationship for Step 3 is:

$$2X_5+4X_6+20X_7+20X_{10}+50X_{11}$$

The prime resource in this step is resource type X_{11}, the engineering staff.

The particular manpower loading relationship for Step 4 is:

$$1X_5+2X_6+40X_7+100X_{11}$$

The first four development steps has had the engineering staff resource type X_{11} as the prime resource. The bulk of the requirements definition, technical feasibility, kernel design, protocol interface, and communication requirements are basic engineering tasks.

The manpower loading relationships for Step 5 is:

$$100X_1+100X_2+100X_3+6X_6+5X_7+120X_{10}+60X_{11}$$

Although the coefficient value (P_{ij}) of resource type X_{11} is greater than any other singular resource type, the prime resource that must be considered here is the programmers. These resource types are prime due to the function of the phase, software development. The training that is accomplished in this step is why the P_{ij} of X_{10} is high.

The manpower loading relationship for Step 6 is:

$$100X_1+100X_2+100X_3+9X_5+12X_6+45X_7+6X_8+60X_{11}$$

The prime resource type in this step is programmers. The bulk of the integration and primary validation/ verification work is accomplished by programmers.

The manpower loading relationship for Step 7 is:

$$10X_1+10X_2+10X_3+6X_5+12X_6+30X_7+3X_8+30X_{11}$$

The prime resource in this step is both the programmers and engineering staff. Formal verification testing

requires both technical skills in order to identify and correct any identified deficiencies.

The final step in the development of software is the documentation step. The manpower loading relationship for Step 8 is:

$$100X_1+100X_2+100X_3+3X_5+6X_6+45X_7+180X_9+90X_{11}$$

The prime resource in this step is X_9, the documentation librarian. This step considers that the documentation librarian resource also develops and conducts the training.

VITA

JOSEPH BRUCE MICHELS

Captain, United States Air Force

PII Redacted [redacted]

Bachelor of Science/Electronic Engineering Technology
Weber State College
Odgen, Utah
June 1976

Master of Science/Systems Management
University of Southern California
Los Angeles, California
June 1980

Current Assignment
Headquarters United States Air Force
Deputy Chief of Staff, Logistics and Engineering
Directorate of Logistics Plans and Programs
Division of Logistics Concepts
Washington, D.C. 20050

[redacted]

The typist for this dissertation was Jimmye Hill